

## Features

- Incorporates the ARM7TDMI<sup>®</sup> ARM<sup>®</sup> Thumb<sup>®</sup> Processor
  - High-performance 32-bit RISC Architecture
  - High-density 16-bit Instruction Set
  - Leader in MIPS/Watt
  - Embedded ICE In-circuit Emulation, Debug Communication Channel Support
- 32 Kbytes of Internal High-speed Flash, Organized in 256 Pages of 128 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: 4 ms, Including Page Auto-erase, Full Erase Time: 10 ms
  - 10,000 Write Cycles, 10-year Data Retention Capability, Sector Lock Capabilities, Security Bit Guaranteeing Code Confidentiality
  - Fast Flash Programming Interface for High Volume Production
- 8 Kbytes of Internal High-speed SRAM, Single-cycle Access at Maximum Speed
- Memory Controller (MC)
  - Embedded Flash Controller, Abort Status and Misalignment Detection
- Reset Controller (RSTC)
  - Based on Power-on Reset and Low-power Factory-calibrated Brownout Detector
  - Allows External Reset Signal Shaping and Reset Source Status
- Clock Generator (CKGR)
  - Low-power RC Oscillator, 3 to 20 MHz On-chip Oscillator and one PLL
- Power Management Controller (PMC)
  - Software Power Optimization Capabilities, Including Slow Clock Mode (Down to 500 Hz) and Idle Mode
  - Three Programmable External Clock Signals
- Advanced Interrupt Controller (AIC)
  - Individually Maskable, Eight-level Priority, Vectored Interrupt Sources
  - One External Interrupt Source and One Fast Interrupt Source, Spurious Interrupt Protected
- Debug Unit (DBGU)
  - 2-wire UART and Support for Debug Communication Channel interrupt, Programmable ICE Access Prevention
- 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 Signals to the System
  - Counter May Be Stopped While the Processor is in Debug State or in Idle Mode
- Real-time Timer (RTT)
  - 32-bit Free-running Counter with Alarm
  - Runs Off the Internal RC Oscillator
- One Parallel Input/Output Controller (PIOA)
  - Twenty-one 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
- Nine Peripheral Data Controller (PDC) Channels
- One Synchronous Serial Controller (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 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
- One Master/Slave Serial Peripheral Interface (SPI)
  - 8- to 16-bit Programmable Data Length, Four External Peripheral Chip Selects
- One Three-channel 16-bit Timer/Counter (TC)
  - Three External Clock Inputs, Two multi-purpose I/O Pins per Channel
  - Double PWM Generation, Capture/Waveform Mode, Up/Down Capability



## AT91 ARM<sup>®</sup> Thumb<sup>®</sup>-based Microcontrollers

### AT91SAM7S32

## Preliminary

6071A-ATARM-28-Oct-04



- One Four-channel 16-bit PWM Controller (PWMC)
- One Two-wire Interface (TWI)
  - Master Mode Support Only, All Two-wire Atmel EEPROMs Supported
- One 8-channel 10-bit Analog-to-Digital Converter, Four Channels Multiplexed with Digital I/Os
- IEEE 1149.1 JTAG Boundary Scan on All Digital Pins
- 5V-tolerant I/Os, including Four High-current Drive I/O lines, Up to 16 mA Each
- Power Supplies
  - Embedded 1.8V Regulator, Drawing up to 100 mA for the Core and External Components
  - 3.3V VDDIO I/O Lines Power Supply, Independent 3.3V VDDFLASH Flash Power Supply
  - 1.8V VDDCORE Core Power Supply with Brownout Detector
- Fully Static Operation: Up to 55 MHz at 1.65V and 85°C Worst Case Conditions
- Available in a 48-lead LQFP Package

## Description

Atmel's AT91SAM7S32 is a member of a series of low pin count Flash microcontrollers based on the 32-bit ARM RISC processor. It features a 32 Kbyte high-speed Flash and an 8 Kbyte SRAM, a large set of peripherals 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.

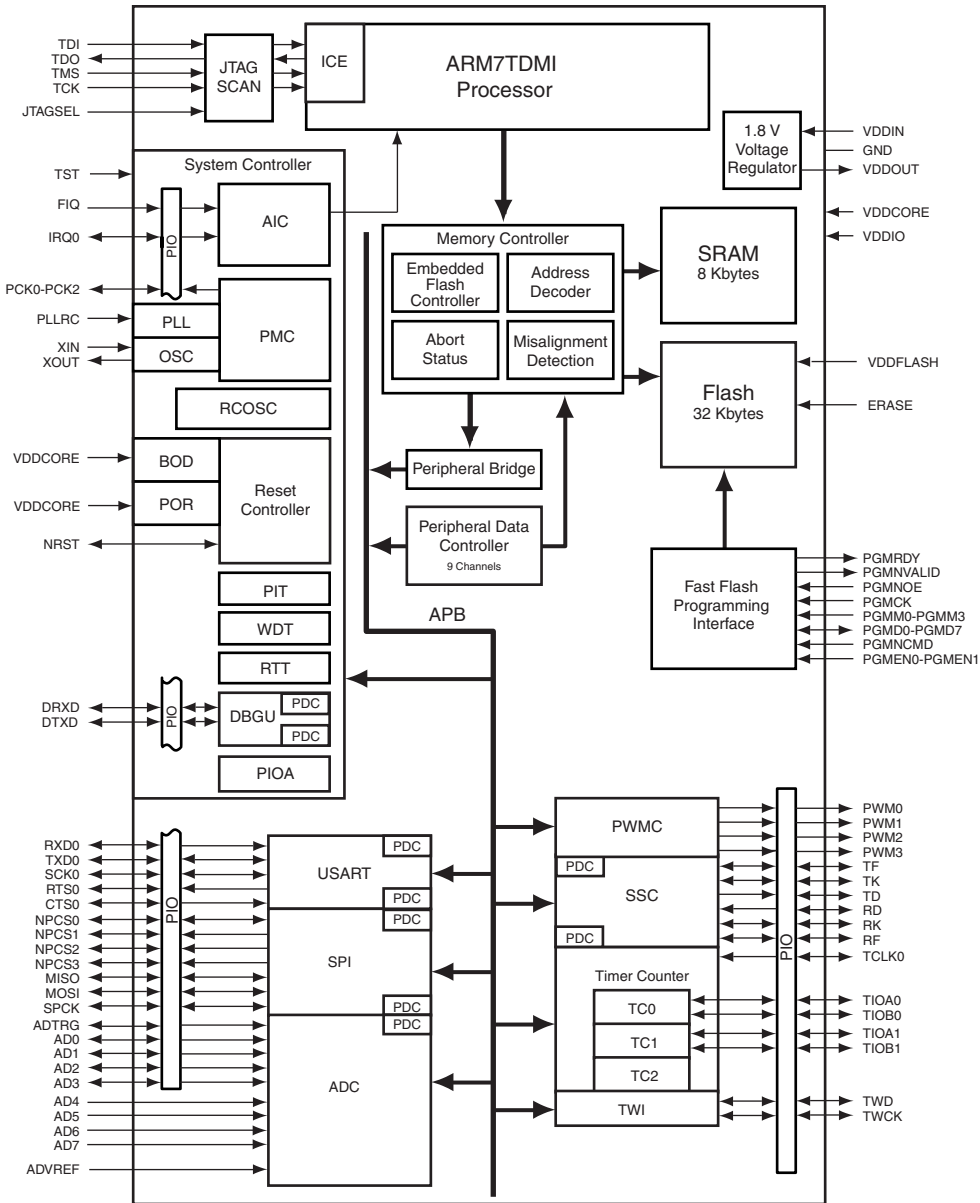
The embedded Flash memory can be programmed in-system via the JTAG-ICE interface or via a parallel interface on a production programmer prior to mounting. Built-in lock bits and a security bit protect the firmware from accidental overwrite and preserves its confidentiality.

The AT91SAM7S32 system controller includes a reset controller capable of managing the power-on sequence of the microcontroller and the complete system. Correct device operation can be monitored by a built-in brownout detector and a watchdog running off an integrated RC oscillator.

The AT91SAM7S32 is a general-purpose microcontroller. Its aggressive price point and high level of integration pushes its scope of use far into the cost-sensitive, high-volume consumer market.

## Block Diagram

Figure 1. AT91SAM7S32 Block Diagram





## Signal Description

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

**Table 1.** Signal Description List

Signal Name	Function	Type	Active Level	Comments
<b>Power</b>				
VDDIN	Main Power Supply Input	Power		3.0V to 3.6V
VDDOUT	Voltage Regulator Output	Power		1.85V nominal
VDDFLASH	Flash Power Supply	Power		3.0V to 3.6V
VDDIO	I/O Lines Power Supply	Power		3.0V to 3.6V
VDDCORE	Core Power Supply	Power		1.65V to 1.95V
VDDPLL	PLL	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 - PCK2	Programmable Clock Output	Output		
<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>Flash Memory</b>				
ERASE	Flash and NVM Configuration Bits Erase Command	Input	High	Pull-down resistor
<b>Reset/Test</b>				
NRST	Microcontroller Reset	I/O	Low	Pull-Up resistor
TST	Test Mode Select	Input		Pull-down resistor
<b>Debug Unit</b>				
DRXD	Debug Receive Data	Input		
DTXD	Debug Transmit Data	Output		
<b>AIC</b>				
IRQ0	External Interrupt Input	Input		
FIQ	Fast Interrupt Input	Input		
<b>PIO</b>				
PA0 - PA20	Parallel IO Controller A	I/O		Pulled-up input at reset
<b>USART</b>				

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

Signal Name	Function	Type	Active Level	Comments
SCK0	Serial Clock	I/O		
TXD0	Transmit Data	I/O		
RXD0	Receive Data	Input		
RTS0	Request To Send	Output		
CTS0	Clear To Send	Input		
<b>Synchronous Serial Controller</b>				
TD	Transmit Data	Output		
RD	Receive Data	Input		
TK	Transmit Clock	I/O		
RK	Receive Clock	I/O		
TF	Transmit Frame Sync	I/O		
RF	Receive Frame Sync	I/O		
<b>Timer/Counter</b>				
TCLK0	External Clock Input	Input		
TIOA0 - TIOA1	I/O Line A	I/O		
TIOB0 - TIOB1	I/O Line B	I/O		
<b>PWM Controller</b>				
PWM0 - PWM3	PWM Channels	Output		
<b>SPI</b>				
MISO	Master In Slave Out	I/O		
MOSI	Master Out Slave In	I/O		
SPCK	SPI Serial Clock	I/O		
NPCS0	SPI Peripheral Chip Select 0	I/O	Low	
NPCS1-NPCS3	SPI Peripheral Chip Select 1 to 3	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>				
AD0-AD3	Analog Inputs	Analog		Digital pulled-up inputs at reset
AD4-AD7	Analog Inputs	Analog		Analog Inputs
ADTRG	ADC Trigger	Input		
ADVREF	ADC Reference	Analog		
<b>Fast Flash Programming Interface</b>				
PGMEN0-PGMEN1	Programming Enabling	Input		
PGMM0-PGMM3	Programming Mode	Input		



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

Signal Name	Function	Type	Active Level	Comments
PGMD0 - PGMD7	Programming Data	I/O		
PGMRDY	Programming Ready	Output	High	
PGMNVALID	Data Direction	Output	Low	
PGMNOE	Programming Read	Input	Low	
PGMCK	Programming Clock	Input		
PGMNCMD	Programming Command	Input	Low	

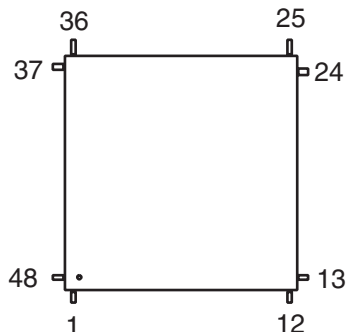
## Package and Pinout

The AT91SAM7S32 is available in a 48-lead LQFP package.

## 48-lead LQFP Mechanical Overview

Figure 2 shows the orientation of the 48-lead LQFP package. A detailed mechanical description is given in the section Mechanical Characteristics of the product datasheet.

**Figure 2.** 48-lead LQFP Package Pinout (Top View)



## Pinout

**Table 2.** AT91SAM7S32 Pinout in 48-lead LQFP Package

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

## Power Considerations

### Power Supplies

The AT91SAM7S32 has six types of power supply pins and integrates a voltage regulator, allowing the device to be supplied with only one voltage. The six power supply pin types are:

- VDDIN pin. It powers the voltage regulator; voltage ranges from 3.0V to 3.6V, 3.3V nominal. If the voltage regulator is not used, VDDIN should be connected to GND.
- VDDOUT pin. It is the output of the 1.8V voltage regulator.
- VDDIO pin. It powers the I/O lines and the USB transceivers; dual voltage range is supported. Ranges from 3.0V to 3.6V, 3.3V nominal.
- VDDFLASH pin. It powers a part of the Flash and is required for the Flash to operate correctly; voltage ranges from 3.0V to 3.6V, 3.3V nominal.
- VDDCORE pins. They power the logic of the device; voltage ranges from 1.65V to 1.95V, 1.8V typical. It can be connected to the VDDOUT pin with decoupling capacitor. VDDCORE is required for the device, including its embedded Flash, to operate correctly.
- VDDPLL pin. It powers the oscillator and the PLL. It can be connected directly to the VDDOUT pin.

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.

### Power Consumption

The AT91SAM7S32 has a static current of less than 60  $\mu$ A on VDDCORE at 25°C, including the RC oscillator, the voltage regulator and the power-on reset when the brownout detector is deactivated. Activating the brownout detector adds 20  $\mu$ A static current.

The dynamic power consumption on VDDCORE is less than 50 mA at full speed when running out of the Flash. Under the same conditions, the power consumption on VDDFLASH does not exceed 10 mA.

### Voltage Regulator

The AT91SAM7S32 embeds a voltage regulator that is managed by the System Controller.

In Normal Mode, the voltage regulator consumes less than 100  $\mu$ A static current and draws 100 mA of output current.

The voltage regulator also has a Low-power Mode. In this mode, it consumes less than 20  $\mu$ A static current and draws 1 mA of output current.

Adequate output supply decoupling is mandatory for VDDOUT 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 VDDOUT and GND as close to the chip as possible. One external 2.2  $\mu$ F (or 3.3  $\mu$ F) X7R capacitor must be connected between VDDOUT and GND.

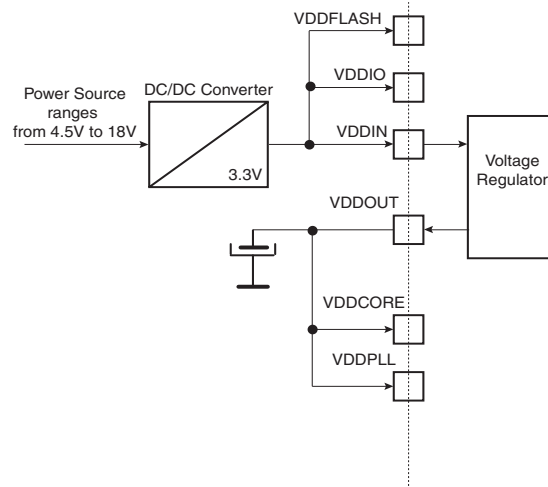
Adequate input supply decoupling is mandatory for VDDIN in order to improve startup stability and reduce source voltage drop. The input decoupling capacitor should be placed close to the chip. For example, two capacitors can be used in parallel: 100 nF NPO and 4.7  $\mu$ F X7R.



## Typical Powering Schematics

The AT91SAM7S32 supports a 3.3V single supply mode. The internal regulator is connected to the 3.3V source and its output feeds VDDCORE and the VDDPLL.

**Figure 3.** 3.3V System Single Power Supply Schematic



## I/O Lines Considerations

### JTAG Port Pins

TMS, TDI and TCK are schmitt trigger inputs. TMS and TCK are 5-V tolerant, TDI is not. TMS, TDI and TCK do not integrate a pull-up resistor.

TDO is an output, driven at up to VDDIO, and has no pull resistor.

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

### Test Pin

The pin TST is used for manufacturing test or fast programming mode of the AT91SAM7S32 when asserted high. The pin TST integrates a permanent pull-down resistor of about 15 kW to GND, so that it can be left unconnected for normal operations.

To enter fast programming mode, the pin TST and the pin PA0 and PA1 should be both tied high.

Driving the pin TST at a high level while PA0 or PA1 is driven at 0 leads to unpredictable results.

### Reset Pin

The pin NRST is bi-directional. 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 pin NRST as system user reset, and the use of the signal NRST to reset all the components of the system.

The pin NRST integrates a permanent pull-up resistor to VDDIO.

### ERASE Pin

The pin ERASE is used to re-initialize the Flash content and some of its NVM bits. It integrates a permanent pull-down resistor of about 15 kW to GND, so that it can be left unconnected for normal operations.

### PIO Controller Lines

All the I/O lines PA0 to PA20 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 VDDIO, but can be driven with a voltage of up to 5.5V. However, driving an I/O line with a voltage over VDDIO while the programmable pull-up resistor is enabled can lead to unpredictable results. Care should be taken, in particular at reset, as all the I/O lines default in input with pull-up resistor enabled at reset.

### I/O Line Drive Levels

The PIO lines PA0 to PA3 are high-drive current capable. Each of these I/O lines can drive up to 16 mA permanently.

The remaining I/O lines can draw only 8 mA.

However, the total current drawn by all the I/O lines cannot exceed 100 mA.

## Processor and Architecture

### ARM7TDMI Processor

- RISC processor based on ARMv4T Von Neumann architecture
  - Runs at up to 55 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)

### Debug and Test Features

- Integrated 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

### Memory Controller

- Bus Arbiter
  - Handles requests from the ARM7TDMI and the Peripheral Data Controller
- Address decoder provides selection signals for
  - Three internal 1 Mbyte 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 SRAM in place of the embedded non-volatile memory
  - Allows handling of dynamic exception vectors
- Embedded Flash Controller
  - Embedded Flash interface, up to three programmable wait states
  - Prefetch buffer, buffering and anticipating the 16-bit requests, reducing the required wait states
  - Key-protected program, erase and lock/unlock sequencer
  - Single command for erasing, programming and locking operations
  - Interrupt generation in case of forbidden operation



## Peripheral Data Controller

- Handles data transfer between peripherals and memories
- Nine channels
  - Two for the USART
  - Two for the Debug Unit
  - Two for the Serial Synchronous Controller
  - Two for the Serial Peripheral Interface
  - One for the 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

## Memories

- 32 Kbytes of Flash Memory
  - 256 pages of 128 bytes
  - Fast access time, 30 MHz single-cycle access in worst case conditions
  - Page programming time: 4 ms, including page auto-erase
  - Page programming without auto-erase: 2 ms
  - Full chip erase time: 10 ms
  - 10,000 write cycles, 10-year data retention capability
  - 8 lock bits, each protecting 8 sectors of 32 pages
  - Protection Mode to secure contents of the Flash
- 8 Kbytes of Fast SRAM
  - Single-cycle access at full speed

## Memory Mapping

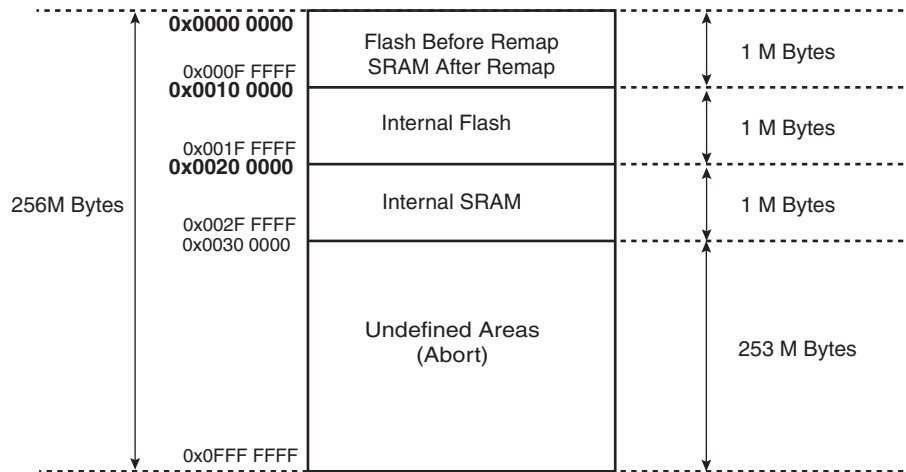
### Internal SRAM

The AT91SAM7S32 embeds a high-speed 8-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.

### Internal Flash

The AT91SAM7S32 features one bank of 32 Kbytes of Flash. At any time, the Flash is mapped to address 0x0010 0000. It is also accessible at address 0x0 after the reset and before the Remap Command.

**Figure 4.** Internal Memory Mapping



## Embedded Flash

### Flash Overview

The Flash of the AT91SAM7S32 is organized in 256 pages of 128 bytes. The 32,768 bytes are organized in 32-bit words.

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

The Flash benefits from the integration of a power reset cell and from the brownout detector. This prevents code corruption during power supply changes, even in the worst conditions.

### 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.

### Lock Regions

The Embedded Flash Controller manages 8 lock bits to protect 8 regions of the flash against inadvertent flash erasing or programming commands. The AT91SAM7S32 contains 8 lock regions and each lock region contains 32 pages of 128 bytes. Each lock region has a size of 4 Kbytes.

If a locked-regions erase or program command occurs, the command is aborted and the EFC trigs an interrupt.

The 8 NVM bits are software programmable through the EFC User Interface. The command "Set Lock Bit" enables the protection. The command "Clear Lock Bit" unlocks the lock region.

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

### Security Bit Feature

The AT91SAM7S32 features a security bit, based on a specific NVM-Bit. When the security is enabled, any access to the Flash, either through the ICE interface or through the Fast Flash Programming Interface, is forbidden. This ensures the confidentiality of the code programmed in the Flash.

This security bit can only be enabled, through the Command "Set Security Bit" of the EFC User Interface. Disabling the security bit can only be achieved by asserting the ERASE pin at 1, and after a full flash erase is performed. When the security bit is deactivated, all accesses to the flash are permitted.

It is important to note that the assertion of the ERASE pin should always be longer than 50 ms.

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

## Non-volatile Brownout Detector Control

Two general purpose NVM (GPNVM) bits are used for controlling the brownout detector (BOD), so that even after a power loss, the brownout detector operations remain as defined by the user.

These two GPNVM bits can be cleared or set respectively through the commands "Clear General-purpose NVM Bit" and "Set General-purpose NVM Bit" of the EFC User Interface.

- GPNVM Bit 0 is used as a brownout detector enable bit. Setting the GPNVM Bit 0 enables the BOD, clearing it disables the BOD. Asserting ERASE clears the GPNVM Bit 0 and thus disables the brownout detector by default.
- The GPNVM Bit 1 is used as a brownout reset enable signal for the reset controller. Setting the GPNVM Bit 1 enables the brownout reset when a brownout is detected, Clearing the GPNVM Bit 1 disables the brownout reset. Asserting ERASE disables the brownout reset by default.

## Calibration Bits

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

## Fast Flash Programming Interface

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

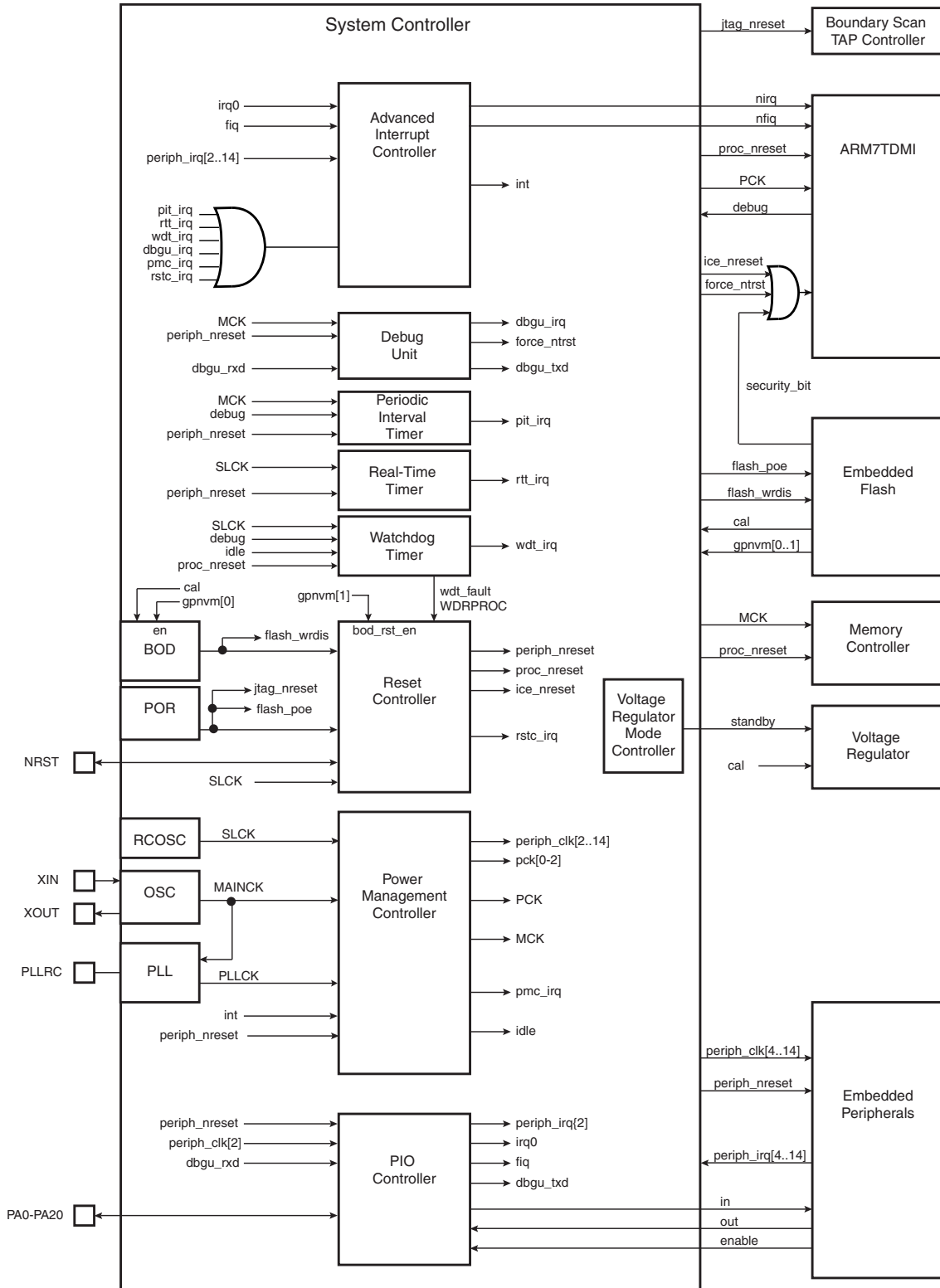
The FFPI supports read, page program, page erase, full erase, lock, unlock and protect commands.

The Fast Flash Programming Interface is enabled and the Fast Programming Mode is entered when the TST pin and the PA0 and PA1 pins are all tied high.

# System Controller

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

Figure 5. System Controller Block Diagram





## System Controller Mapping

The System Controller peripherals are all mapped to the highest 4 Kbytes of address space, between addresses 0xFFFF F000 and 0xFFFF FFFF.

Figure 6 shows the mapping of the System Controller. Note that the Memory Controller configuration user interface is also mapped within this address space.

**Figure 6.** System Controller Mapping

Address	Peripheral	Peripheral Name	Size
0xFFFF F000	AIC	Advanced Interrupt Controller	512 Bytes/128 registers
0xFFFF F1FF			
0xFFFF F200	DBGU	Debug Unit	512 Bytes/128 registers
0xFFFF F3FF			
0xFFFF F400	PIOA	PIO Controller A	512 Bytes/128 registers
0xFFFF F5FF			
0xFFFF F600	Reserved		
0xFFFF FBFF			
0xFFFF FC00	PMC	Power Management Controller	256 Bytes/64 registers
0xFFFF FCFF			
0xFFFF FD00	RSTC	Reset Controller	16 Bytes/4 registers
0xFFFF FD0F	Reserved		
0xFFFF FD20	RTT	Real-time Timer	16 Bytes/4 registers
0xFFFF FC2F	PIT	Periodic Interval Timer	16 Bytes/4 registers
0xFFFF FD30			
0xFFFF FC3F	WDT	Watchdog Timer	16 Bytes/4 registers
0xFFFF FD40			
0xFFFF FD4F	Reserved		
0xFFFF FD60	VREG	Voltage Regulator Mode Controller	4 Bytes/1 register
0xFFFF FC6F	Reserved		
0xFFFF FD70			
0xFFFF FEFF	MC	Memory Controller	256 Bytes/64 registers
0xFFFF FF00			
0xFFFF FFFF			

## Reset Controller

The Reset Controller is based on a power-on reset cell and one brownout detector. It gives the status of the last reset, indicating whether it is a power-up reset, a software reset, a user reset, a watchdog reset or a brownout reset. In addition, it controls the internal resets and the NRST pin output. It allows to shape a signal on the NRST line, guaranteeing that the length of the pulse meets any requirement.

## Brownout Detector and Power-on Reset

The AT91SAM7S32 embeds a brownout detection circuit and a power-on reset cell. Both are supplied with and monitor VDDCORE. Both signals are provided to the Flash to prevent any code corruption during power-up or power-down sequences or if brownouts occur on the VDDCORE power supply.

The power-on reset cell has a limited-accuracy threshold at around 1.5V. Its output remains low during power-up until VDDCORE goes over this voltage level. This signal goes to the reset controller and allows a full re-initialization of the device.

The brownout detector monitors the VDDCORE level during operation by comparing it to a fixed trigger level. It secures system operations in the most difficult environments and prevents code corruption in case of brownout on the VDDCORE.

Only VDDCORE is monitored, as a voltage drop on VDDFLASH or any other power supply of the device cannot affect the Flash.

When the brownout detector is enabled and VDDCORE decreases to a value below the trigger level ( $V_{bot-}$ , defined as  $V_{bot} - hyst/2$ ), the brownout output is immediately activated.

When VDDCORE increases above the trigger level ( $V_{bot+}$ , defined as  $V_{bot} + hyst/2$ ), the reset is released. The brownout detector only detects a drop if the voltage on VDDCORE stays below the threshold voltage for longer than about 1 $\mu$ s.

The threshold voltage has a hysteresis of about 50 mV, to ensure spike free brownout detection. The typical value of the brownout detector threshold is 1.68V with an accuracy of  $\pm 2\%$  and is factory calibrated.

The brownout detector is low-power, as it consumes less than 20  $\mu$ A static current. However, it can be deactivated to save its static current. In this case, it consumes less than 1 $\mu$ A. The deactivation is configured through the GPNVM bit 0 of the Flash.

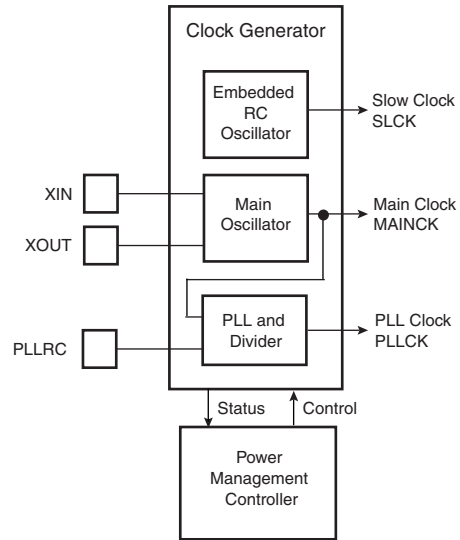
## Clock Generator

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 200 MHz

It provides SLCK, MAINCK and PLLCK.

**Figure 7.** Clock Generator Block Diagram



## Power Management Controller

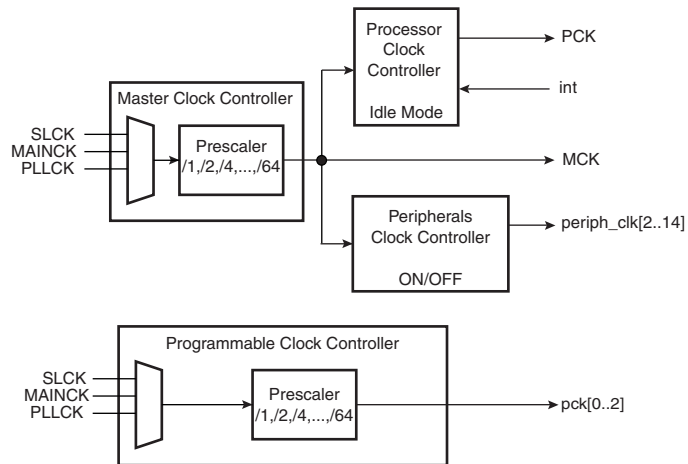
The Power Management Controller uses the Clock Generator outputs to provide:

- the Processor Clock PCK
- the Master Clock MCK
- all the peripheral clocks, independently controllable
- three 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, thus allowing reduced power consumption while waiting for an interrupt.

**Figure 8.** Power Management Controller Block Diagram



## Advanced Interrupt Controller

- Controls the interrupt lines (nIRQ and nFIQ) of an 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 (RTT, PIT, EFC, 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
- 8-level Priority Controller
  - Drives the normal interrupt of the processor
  - Handles priority of the interrupt sources
  - Higher priority interrupts can be served during service of 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

## Debug Unit

- Comprises:
  - One two-pin UART
  - One Interface for the Debug Communication Channel (DCC) support
  - One set of Chip ID Registers
  - One interface providing ICE Access Prevention
- Two-pin UART
  - Implemented features are compatible with the USART
  - 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 0x27080340 (VERSION 0)

## Periodic Interval Timer

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

## 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

## Real-time Timer

- 32-bit free-running counter with alarm running on prescaled SLCK
- Programmable 16-bit prescaler for SLCK accuracy compensation

## PIO Controller

- One PIO Controller, controlling 21 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

## Voltage Regulator Controller

The aim of this controller is to select the Power Mode of the Voltage Regulator between Normal Mode (bit 0 is cleared) or Standby Mode (bit 0 is set).

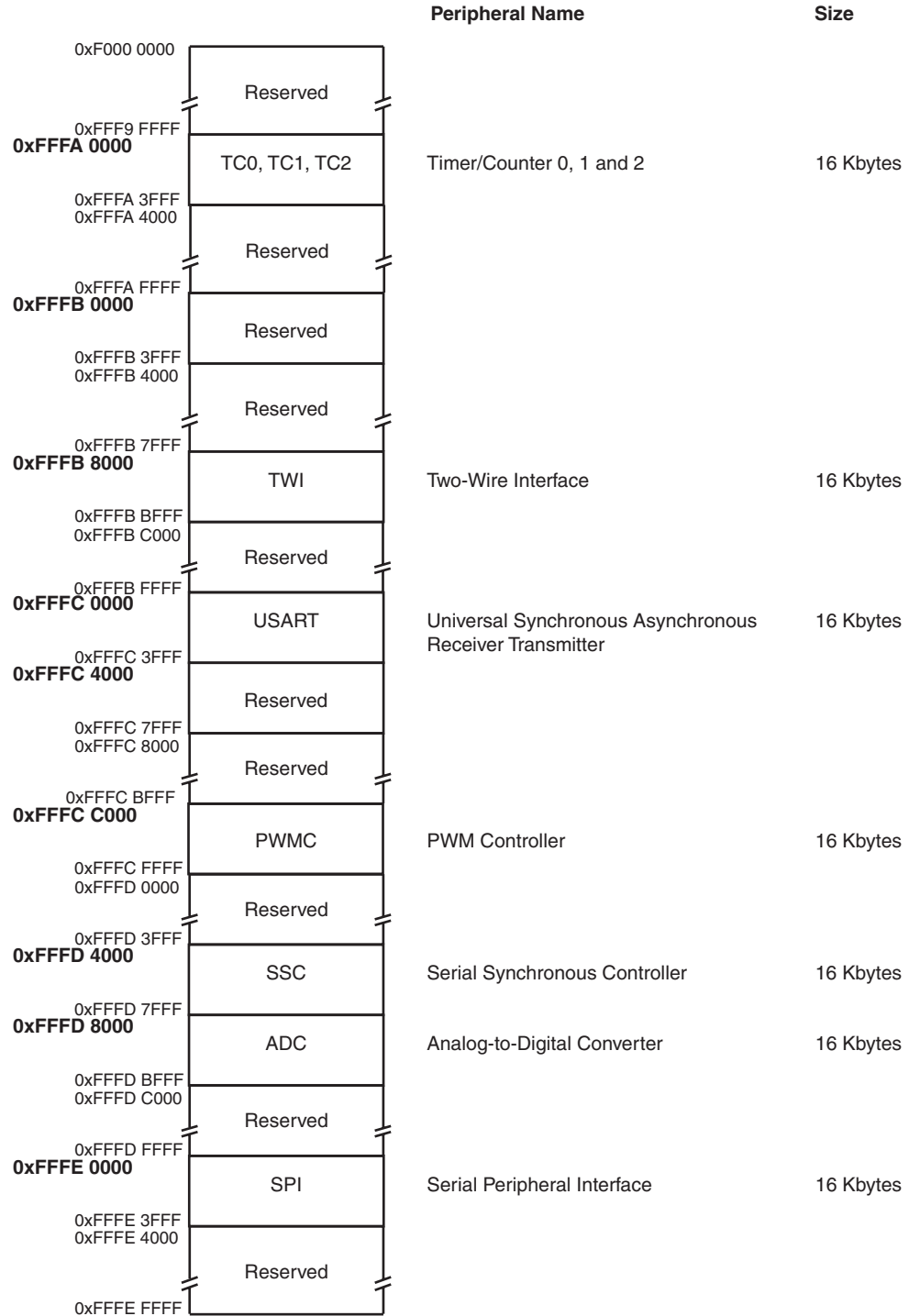


# Peripherals

## Peripheral Mapping

Each peripheral is allocated 16 Kbytes of address space.

**Figure 9.** User Peripheral Mapping



## Peripheral Multiplexing on PIO Lines

The AT91SAM7S32 features one PIO controller, PIOA, that multiplexes the I/O lines of the peripheral set.

PIO Controller A controls 21 lines. Each line can be assigned to one of two peripheral functions, A or B. Some of them can also be multiplexed with the analog inputs of the ADC Controller.

Table 3 on page 23 defines how the I/O lines of the peripherals A, B or the analog inputs are multiplexed on PIO Controller A. 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 in the table.

All pins reset in their Parallel I/O lines function are configured in input with the programmable pull-up enabled, so that the device is maintained in a static state as soon as a reset is detected.

## PIO Controller A Multiplexing

**Table 3.** Multiplexing on PIO Controller A

PIO Controller A				Application Usage	
I/O Line	Peripheral A	Peripheral B	Comments	Function	Comments
PA0	PWM0	TIOA0	High-Drive		
PA1	PWM1	TIOB0	High-Drive		
PA2	PWM2	SCK0	High-Drive		
PA3	TWD	NPCS3	High-Drive		
PA4	TWCK	TCLK0			
PA5	RXD0	NPCS3			
PA6	TXD0	PCK0			
PA7	RTS0	PWM3			
PA8	CTS0	ADTRG			
PA9	DRXD	NPCS1			
PA10	DTXD	NPCS2			
PA11	NPCS0	PWM0			
PA12	MISO	PWM1			
PA13	MOSI	PWM2			
PA14	SPCK	PWM3			
PA15	TF	TIOA1			
PA16	TK	TIOB1			
PA17	TD	PCK1	AD0		
PA18	RD	PCK2	AD1		
PA19	RK	FIQ	AD2		
PA20	RF	IRQ0	AD3		

## Peripheral Identifiers

The AT91SAM7S32 embeds a wide range of peripherals. Table 4 defines the Peripheral Identifiers of the AT91SAM7S32. A peripheral identifier is required for the control of the peripheral interrupt with the Advanced Interrupt Controller and for the control of the peripheral clock with the Power Management Controller.

**Table 4.** Peripheral Identifiers

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
0	AIC	Advanced Interrupt Controller	FIQ
1	SYSIRQ <sup>(1)</sup>	System Interrupt	
2	PIOA	Parallel I/O Controller A	
3	Reserved		
4	ADC <sup>(1)</sup>	Analog-to Digital Converter	
5	SPI	Serial Peripheral Interface	
6	US	USART	
7	Reserved		
8	SSC	Synchronous Serial Controller	
9	TWI	Two-wire Interface	
10	PWMC	PWM Controller	
11	Reserved		
12	TC0	Timer/Counter 0	
13	TC1	Timer/Counter 1	
14	TC2	Timer/Counter 2	
15 - 29	Reserved		
30	AIC	Advanced Interrupt Controller	IRQ0
31	Reserved		

Note: 1. Setting SYSIRQ and ADC bits in the clock set/clear registers of the PMC has no effect. The System Controller is continuously clocked. The ADC clock is automatically started for the first conversion. In Sleep Mode the ADC clock is automatically stopped after each conversion.

## 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 between consecutive transfers and between clock and data per chip select
  - Programmable delay between consecutive transfers
  - Selectable mode fault detection
  - Maximum frequency at up to Master Clock



## 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

## 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
  - 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
  - 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

## 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

## Timer Counter

- Three 16-bit Timer Counter Channels
  - Three output compare or two input capture
- 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 5

**Table 5.** Timer Counter Clocks 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

## PWM Controller

- Four channels, one 16-bit counter per channel
- Common clock generator, providing thirteen different clocks
  - One 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

## Analog-to-digital Converter

- 8-channel ADC
- 10-bit 100 Ksamples/sec. Successive Approximation Register ADC
- -2/+2 LSB Integral Non Linearity, -1/+2 LSB Differential Non Linearity
- Integrated 8-to-1 multiplexer, offering eight independent 3.3V analog inputs
- External voltage reference for better accuracy on low voltage inputs
- Individual enable and disable of each channel
- Multiple trigger source
  - Hardware or software trigger
  - External trigger pin
  - Timer Counter 0 to 1 outputs TIOA0 to TIOA1 trigger
- Sleep Mode and conversion sequencer
  - Automatic wakeup on trigger and back to sleep mode after conversions of all enabled channels
- Four of eight analog inputs shared with digital signals

## ARM7TDMI Processor Overview

### Overview

The ARM7TDMI core executes both the 32-bit ARM<sup>®</sup> and 16-bit Thumb<sup>®</sup> 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<sup>®</sup> High-performance 32-bit Instruction Set
  - Thumb<sup>®</sup> High Code Density 16-bit Instruction Set
- Three-Stage Pipeline Architecture
  - Instruction Fetch (F)
  - Instruction Decode (D)
  - Execute (E)

## 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)

### Instruction Type

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

### 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.

### 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.

### 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 6.** 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.

## 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.

## 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.

## 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)

## 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 7 gives the ARM instruction mnemonic list.

**Table 7. ARM Instruction Mnemonic List**

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	CDP	Coprocessor Data Processing
ADD	Add	MVN	Move Not
SUB	Subtract	ADC	Add with Carry
RSB	Reverse Subtract	SBC	Subtract with Carry
CMP	Compare	RSC	Reverse Subtract with Carry
TST	Test	CMN	Compare Negated
AND	Logical AND	TEQ	Test Equivalence
EOR	Logical Exclusive OR	BIC	Bit Clear
MUL	Multiply	ORR	Logical (inclusive) OR
SMULL	Sign Long Multiply	MLA	Multiply Accumulate
SMLAL	Signed Long Multiply Accumulate	UMULL	Unsigned Long Multiply
MSR	Move to Status Register	UMLAL	Unsigned Long Multiply Accumulate
B	Branch	MRS	Move From Status Register
BX	Branch and Exchange	BL	Branch and Link
LDR	Load Word	SWI	Software Interrupt
LDRSH	Load Signed Halfword	STR	Store Word
LDRSB	Load Signed Byte	STRH	Store Half Word
LDRH	Load Half Word	STRB	Store Byte
LDRB	Load Byte	STRBT	Store Register Byte with Translation
LDRBT	Load Register Byte with Translation	STRT	Store Register with Translation
LDRT	Load Register with Translation	STM	Store Multiple
LDM	Load Multiple	SWPB	Swap Byte
SWP	Swap Word	MRC	Move From Coprocessor
MCR	Move To Coprocessor	STC	Store From Coprocessor
LDC	Load To Coprocessor		

## 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 8 gives the Thumb instruction mnemonic list.

**Table 8.** Thumb Instruction Mnemonic List

Mnemonic	Operation
MOV	Move
ADD	Add
SUB	Subtract
CMP	Compare
TST	Test
AND	Logical AND
EOR	Logical Exclusive OR
LSL	Logical Shift Left
ASR	Arithmetic Shift Right
MUL	Multiply
B	Branch
BX	Branch and Exchange
LDR	Load Word
LDRH	Load Half Word
LDRB	Load Byte
LDRSH	Load Signed Halfword
LDMIA	Load Multiple
PUSH	Push Register to stack

Mnemonic	Operation
MVN	Move Not
ADC	Add with Carry
SBC	Subtract with Carry
CMN	Compare Negated
NEG	Negate
BIC	Bit Clear
ORR	Logical (inclusive) OR
LSR	Logical Shift Right
ROR	Rotate Right
BL	Branch and Link
SWI	Software Interrupt
STR	Store Word
STRH	Store Half Word
STRB	Store Byte
LDRSB	Load Signed Byte
STMIA	Store Multiple
POP	Pop Register from stack



## AT91SAM7S32 Debug and Test Features

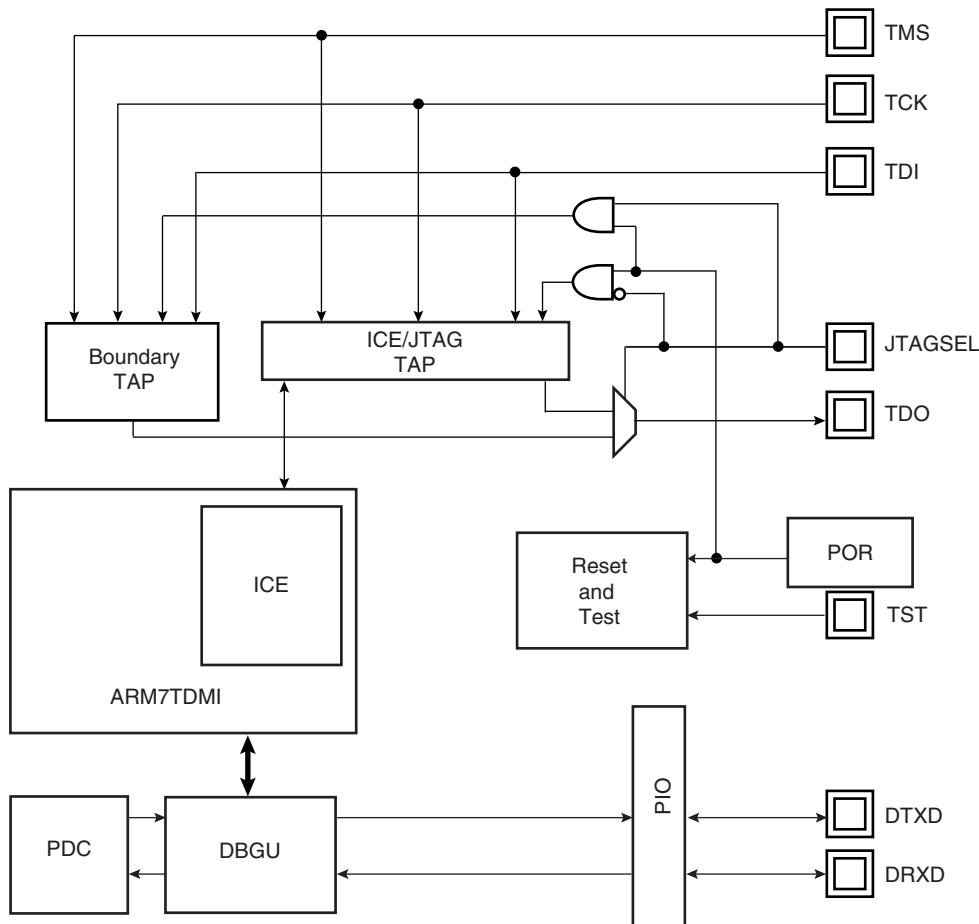
### Description

The AT91SAM7S32 features a number of complementary debug and test capabilities. A common JTAG/ICE (In-Circuit Emulator) 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.

### Block Diagram

Figure 10. Debug and Test Block Diagram

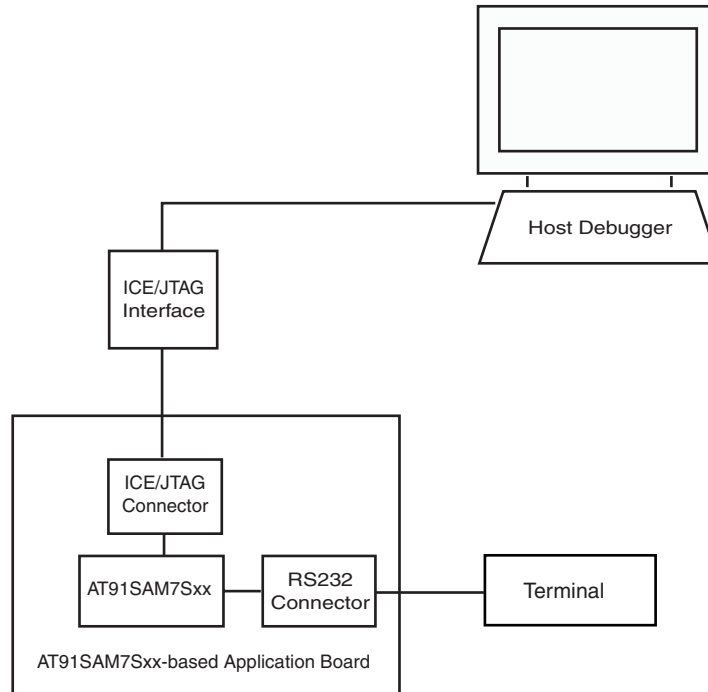


## Application Examples

### Debug Environment

Figure 11 on page 34 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.

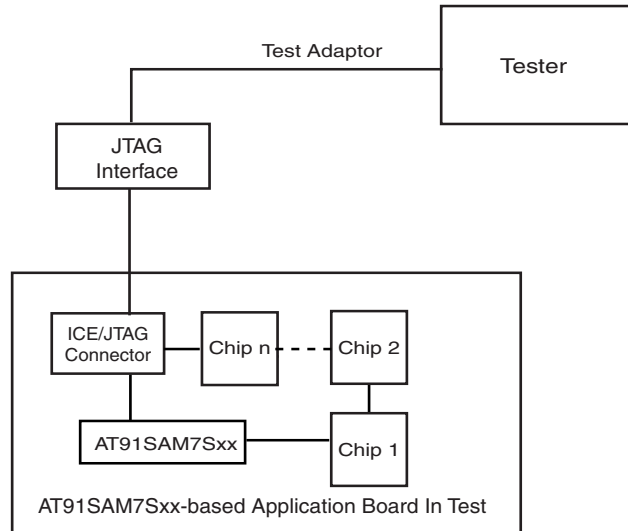
**Figure 11.** Application Debug Environment Example



## Test Environment

Figure 12 on page 35 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 12.** Application Test Environment Example



## Debug and Test Pin Description

**Table 9.** 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	

## Functional Description

### 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.

### Embedded In-circuit Emulator

The ARM7TDMI embedded In-circuit Emulator 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 In-Circuit-Emulator, see the ARM7TDMI (Rev4) Technical Reference Manual (DDI0210B).

### 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 AT91SAM7S32 Debug Unit Chip ID value is 0x27080340 on 32-bit width.

For further details on the Debug Unit, see “Debug Unit (DBGU)” on page 173.

### 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.

## JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains 96 bits that correspond to active pins and associated control signals.

Each AT91SAM7S32 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 10.** AT91SAM7S32 JTAG Boundary Scan Register

Bit Number	Pin Name	Pin Type	Associated BSR Cells
96	PA17/PGMD5/AD0	IN/OUT	INPUT
95			OUTPUT
94			CONTROL
93	PA18/PGMD6/AD1	IN/OUT	INPUT
92			OUTPUT
91			CONTROL
90		internal	
89			
88			
87	PA19/PGMD7/AD2	IN/OUT	INPUT
86			OUTPUT
85			CONTROL
84	PA20/PGMD8/AD3	IN/OUT	INPUT
83			OUTPUT
82			CONTROL
81	PA16/PGMD4	IN/OUT	INPUT
80			OUTPUT
79			CONTROL
78	PA15/PGM3	IN/OUT	INPUT
77			OUTPUT
76			CONTROL
75	PA14/PGMD2	IN/OUT	INPUT
74			OUTPUT
73			CONTROL
72	PA13/PGMD1	IN/OUT	INPUT
71			OUTPUT
70			CONTROL

**Table 10. AT91SAM7S32 JTAG Boundary Scan Register (Continued)**

69			
68		internal	
67			
66			
65		internal	
64			
63			
62		internal	
61			
60			INPUT
59	PA12/PGMD0	IN/OUT	OUTPUT
58			CONTROL
57			INPUT
56	PA11/PGMM3	IN/OUT	OUTPUT
55			CONTROL
54			INPUT
53	PA10/PGMM2	IN/OUT	OUTPUT
52			CONTROL
51			INPUT
50	PA9/PGMM1	IN/OUT	OUTPUT
49			CONTROL
48			INPUT
47	PA8/PGMM0	IN/OUT	OUTPUT
46			CONTROL
45			INPUT
44	PA7/PGMNVALID	IN/OUT	OUTPUT
43			CONTROL
42			INPUT
41	PA6/PGMNOE	IN/OUT	OUTPUT
40			CONTROL
39			INPUT
38	PA5/PGMRDY	IN/OUT	OUTPUT
37			CONTROL
36			INPUT
35	PA4/PGMNCMD	IN/OUT	OUTPUT
34			CONTROL

**Table 10.** AT91SAM7S32 JTAG Boundary Scan Register (Continued)

33			
32		internal	
31			
30			
29		internal	
28			
27			
26		internal	
25			
24			
23		internal	
22			
21			INPUT
20	PA3	IN/OUT	OUTPUT
19			CONTROL
18			INPUT
17	PA2	IN/OUT	OUTPUT
16			CONTROL
15			INPUT
14	PA1/PGMEN1	IN/OUT	OUTPUT
13			CONTROL
12			INPUT
11	PA0/PGMEN0	IN/OUT	OUTPUT
10			CONTROL
9			
8		internal	
7			
6			
5		internal	
4			
3			
2		internal	
1			
0	ERASE	IN	INPUT



## 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

Set to 0x5B07.

### MANUFACTURER IDENTITY[11:1]

Set to 0x01F.

### Bit[0] Required by IEEE Std. 1149.1.

Set to 0x1.

JTAG ID Code value is 05B0\_703F.



## Reset Controller (RSTC)

### Overview

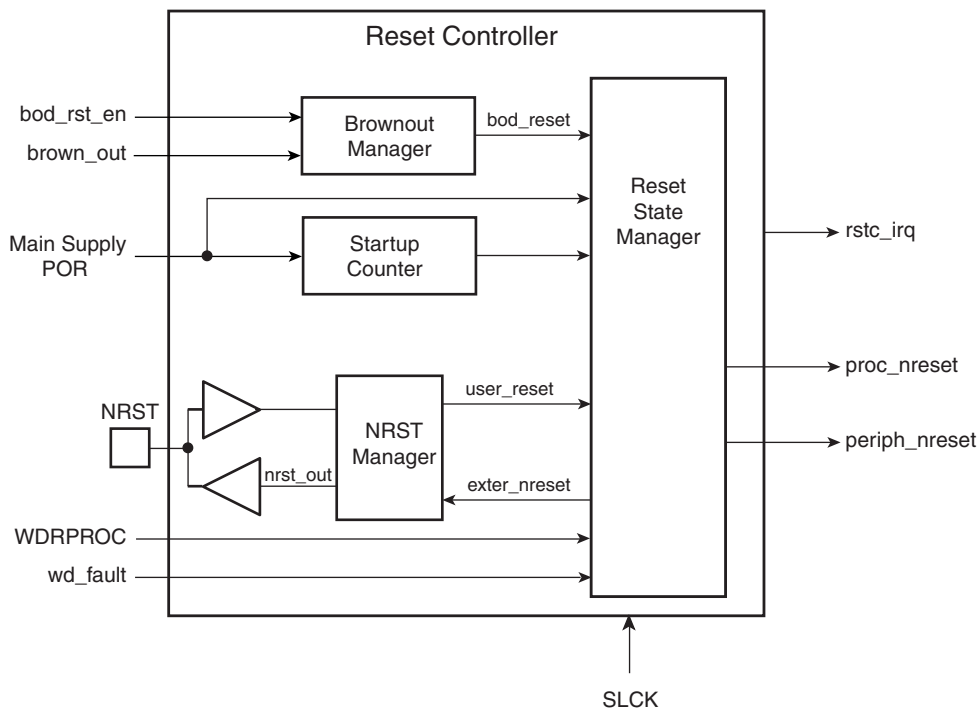
The Reset Controller (RSTC), based on power-on reset cells, handles all the resets of the system without any external components. It reports which reset occurred last.

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

A brownout detection is also available to prevent the processor from falling into an unpredictable state.

### Block Diagram

Figure 13. Reset Controller Block Diagram



## Functional Description

The Reset Controller is made up of an NRST Manager, a Brownout 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.
- `periph_nreset`: Affects the whole set of embedded peripherals.
- `nrst_out`: Drives the NRST pin.

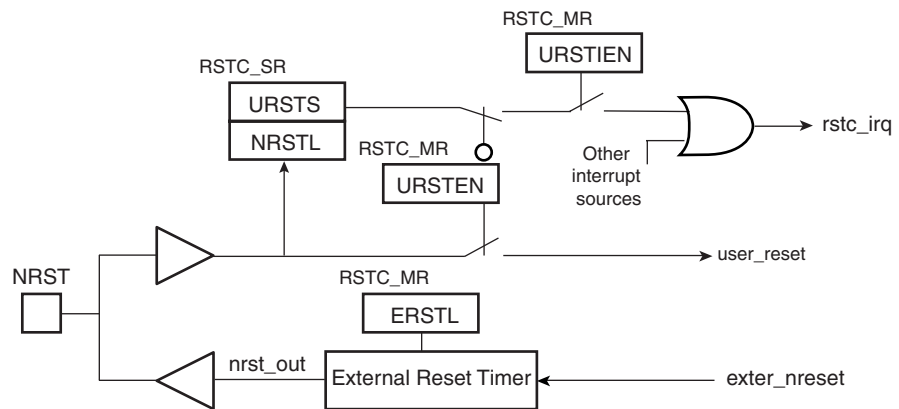
These reset signals are asserted by the Reset Controller, either on external events or on software action. The Reset State Manager controls the generation of reset signals and provides a signal to the NRST Manager when an assertion of the NRST pin is required.

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

## NRST Manager

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

**Figure 14.** NRST Manager



## 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.

## 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.

## Brownout Manager

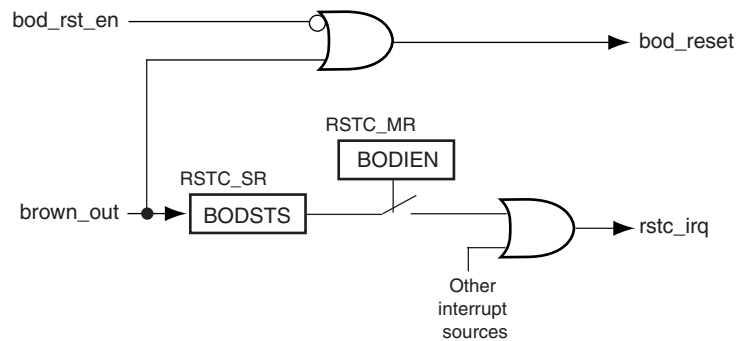
Brownout detection prevents the processor from falling into an unpredictable state if the power supply drops below a certain level. When VDDCORE drops below the brownout threshold, the brownout manager requests a brownout reset by asserting the `bod_reset` signal.

The programmer can disable the brownout reset by setting low the `bod_rst_en` input signal, i.e.; by locking the corresponding general-purpose NVM bit in the Flash. When the brownout reset is disabled, no reset is performed. Instead, the brownout detection is reported in the bit `BODSTS` of `RSTC_SR`. `BODSTS` is set and clears only when `RSTC_SR` is read.

The bit `BODSTS` can trigger an interrupt if the bit `BODIEN` is set in the `RSTC_MR`.

At factory, the brownout reset is disabled.

**Figure 15.** Brownout Manager



## 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.

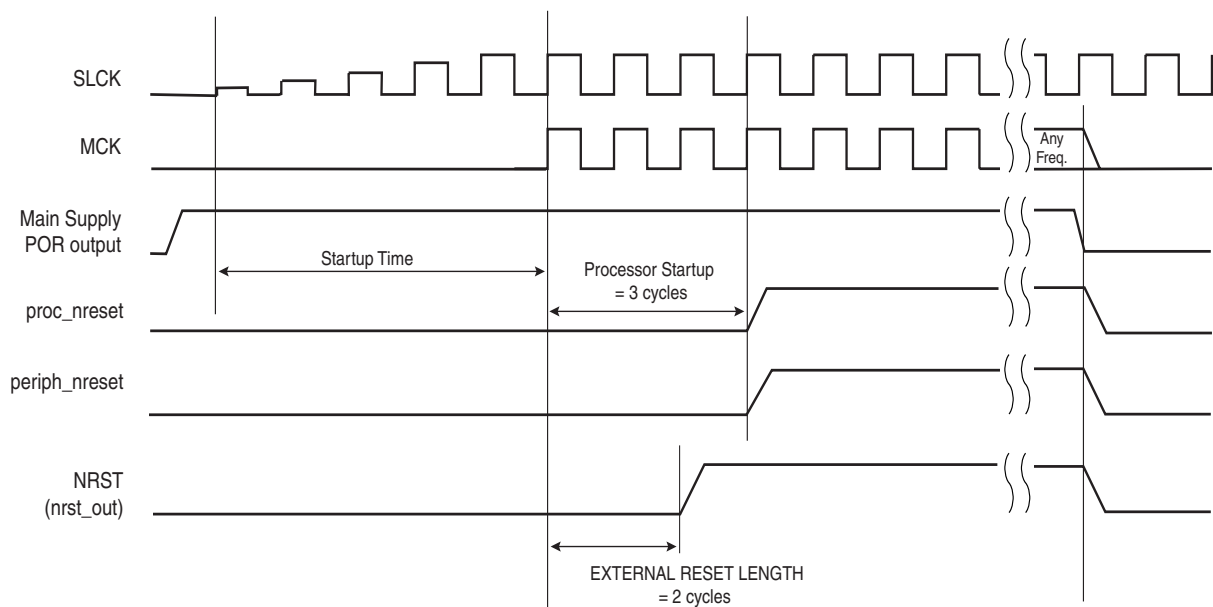
## Power-up Reset

When VDDCORE is powered on, the Main Supply POR cell output is filtered with a start-up counter that operates at Slow Clock. The purpose of this counter is to ensure that the Slow Clock oscillator is stable before starting up the device.

The startup time, as shown in Figure 16, is hardcoded to comply with the Slow Clock Oscillator startup time. After the startup time, the reset signals are released and the field RSTTYP in RSTC\_SR reports a Power-up Reset.

When VDDCORE is detected low by the Main Supply POR Cell, all reset signals are asserted immediately.

**Figure 16. Power-up Reset**



## 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.

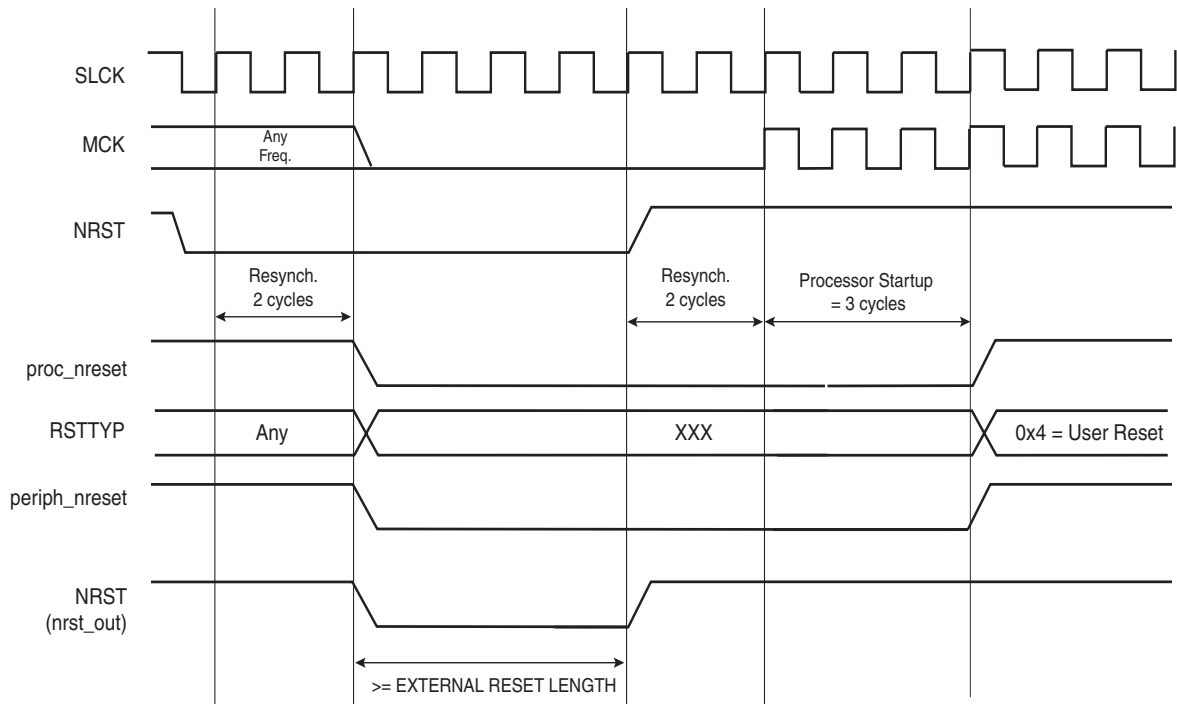
The User Reset is entered as soon as a low level is detected on NRST. The Processor Reset and the Peripheral Reset are asserted.

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 17. User Reset State**



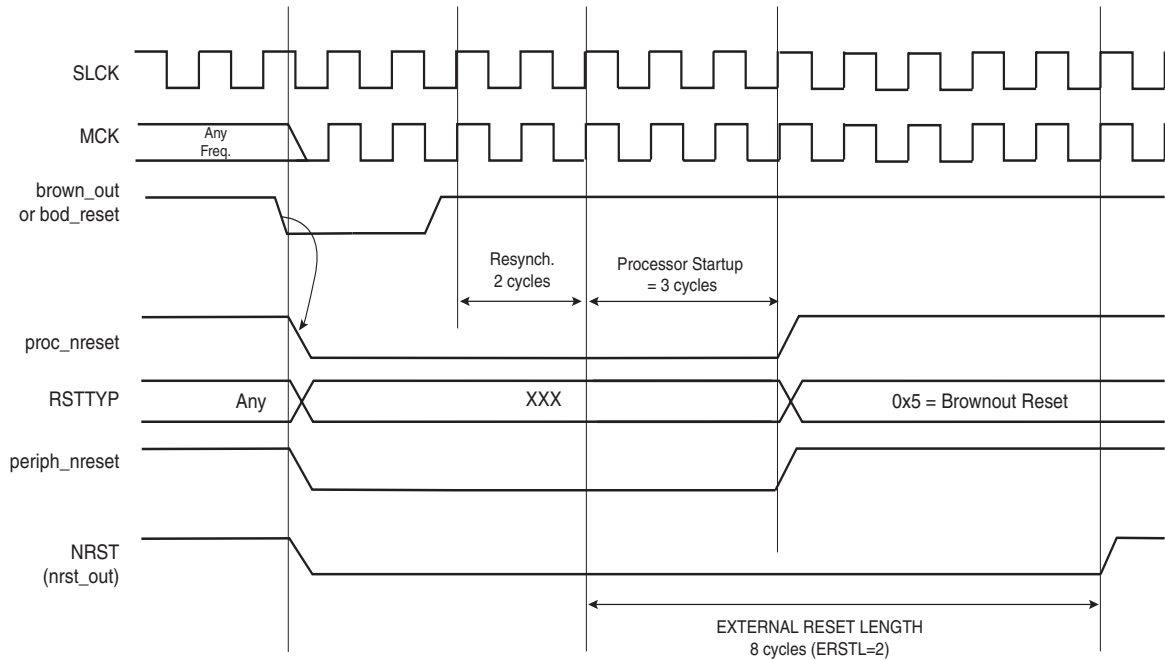
### Brownout Reset

When the brown\_out/bod\_reset signal is asserted, the Reset State Manager immediately enters the Brownout Reset. In this state, the processor, the peripheral and the external reset lines are asserted.

The Brownout Reset is left 3 Slow Clock cycles after the rising edge of brown\_out/bod\_reset after a two-cycle resynchronization. An external reset is also triggered.

When the processor reset is released, the field RSTTYP in RSTC\_SR is loaded with the value 0x5, thus indicating that the last reset is a Brownout Reset.

**Figure 18. Brownout Reset State**



## 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.

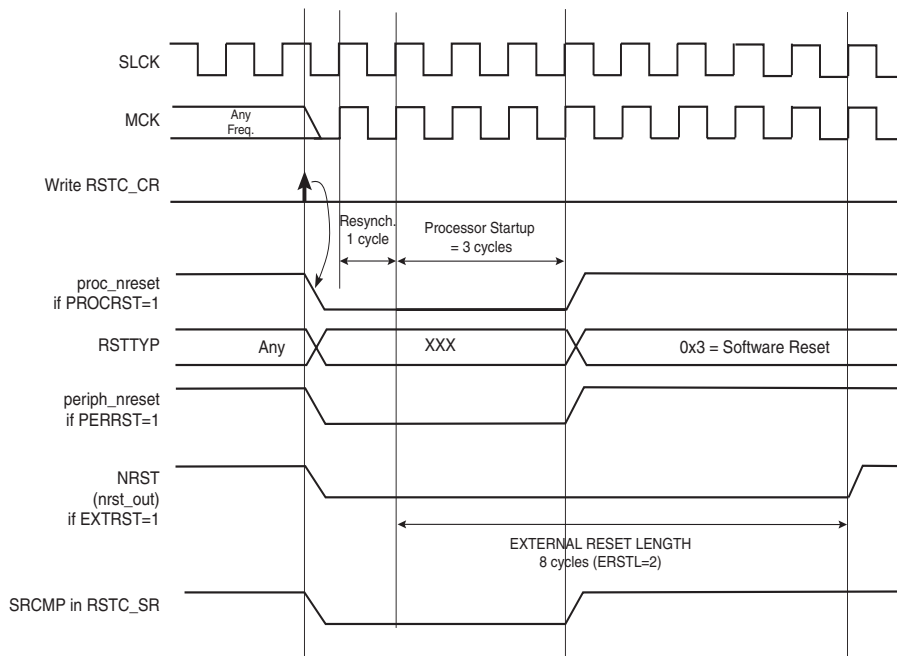
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e.; synchronously to SLCK.

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 19. Software Reset**



## Watchdog Reset

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

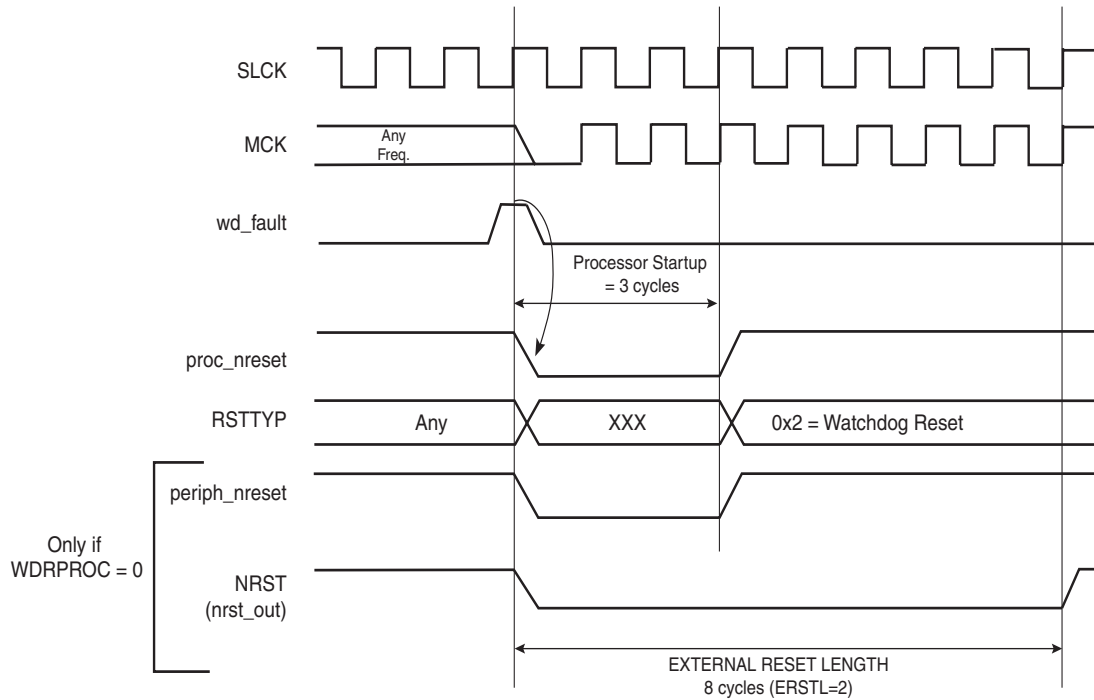
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in WDT\_MR:

- 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 20. Watchdog Reset**





## Reset State Priorities

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

- Power-up Reset
- Brownout Reset
- Watchdog Reset
- Software Reset
- User Reset

Particular cases are listed below:

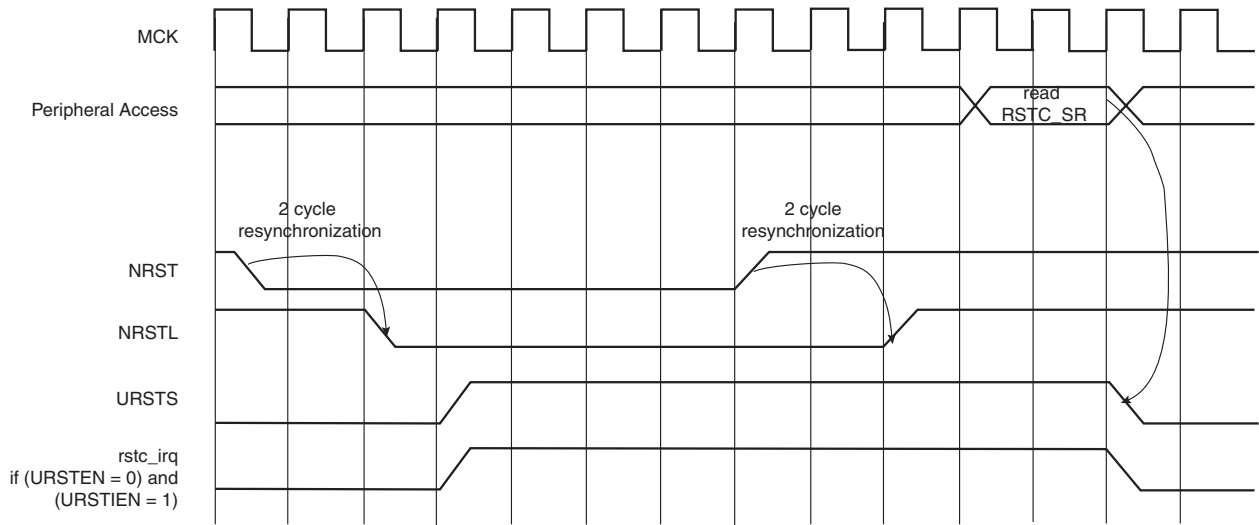
- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the `proc_nreset` signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The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

## 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 21). 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.
- `BODSTS` bit: This bit indicates a brownout detection when the brownout reset is disabled (`bod_rst_en = 0`). It triggers an interrupt if the bit `BODIEN` in the `RSTC_MR` register enables the interrupt. Reading the `RSTC_SR` register resets the `BODSTS` bit and clears the interrupt.

**Figure 21. Reset Controller Status and Interrupt**



**Reset Controller (RSTC) User Interface****Table 11.** Reset Controller Registers

<b>Offset</b>	<b>Register</b>	<b>Name</b>	<b>Access</b>	<b>Reset Value</b>
0x00	Control Register	RSTC_CR	Write-only	-
0x04	Status Register	RSTC_SR	Read-only	0x0000_0000
0x08	Mode Register	RSTC_MR	Read/Write	0x0000_0000

## 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.

## 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
–	–	–	–	–	–	BODSTS	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.

- **BODSTS: Brownout Detection Status**

0 = No brownout high-to-low transition happened since the last read of RSTC\_SR.

1 = A brownout high-to-low transition 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	Power-up Reset	VDDCORE 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
1	0	1	Brownout Reset	BrownOut reset occurred

- **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.

## 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
–	–	–	–	–	–	–	BODIEN
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.

- **BODIEN: Brownout Detection Interrupt Enable**

0 = BODSTS bit in RSTC\_SR at 1 has no effect on rstc\_irq.

1 = BODSTS bit in RSTC\_SR at 1 asserts rstc\_irq.

- **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.

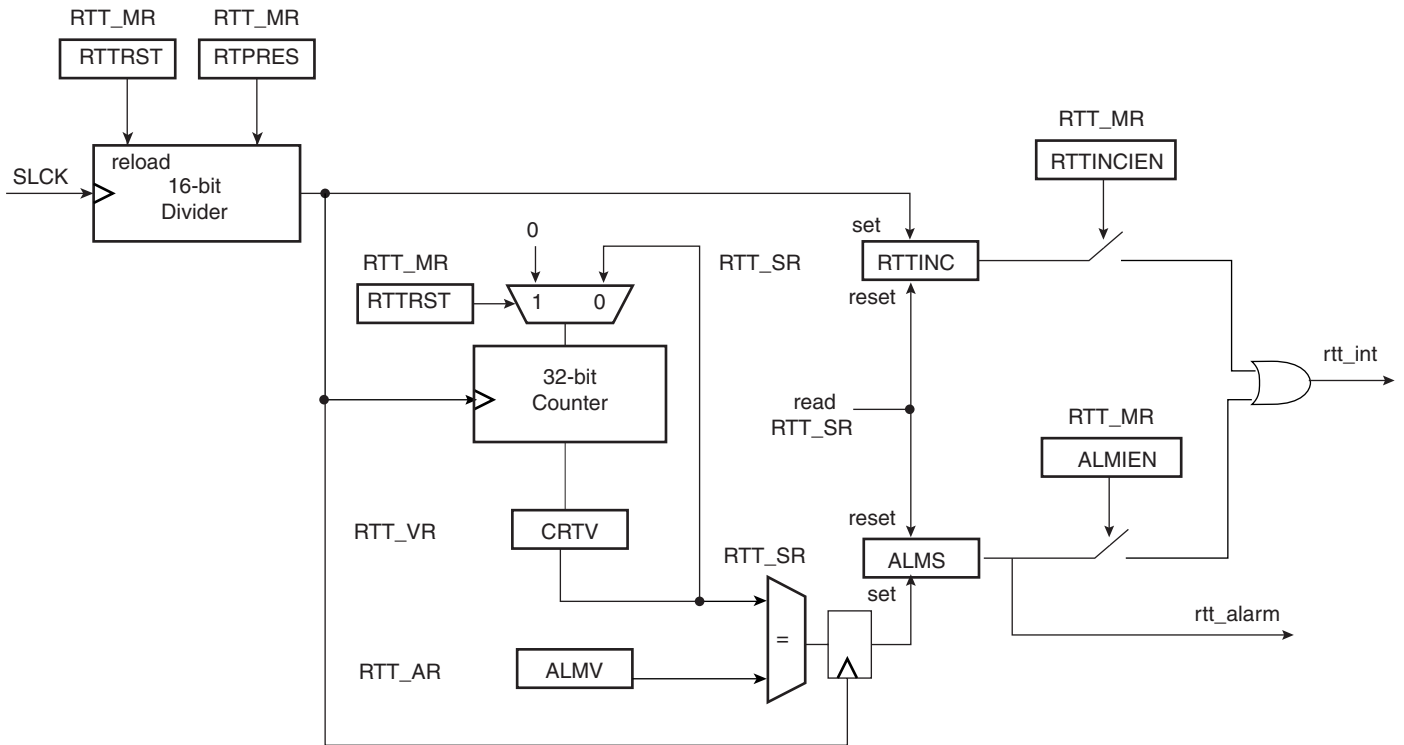
# Real-time Timer (RTT)

## 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.

## Block Diagram

Figure 22. Real-time Timer



## 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 reached by writing RTPRES at 1. In this case, the period of the signal provided to the Real-time Timer counter is 30.52  $\mu$ s (when Slow Clock is 32.768 Hz) and the maximum the Real-time Timer can cover is 131072 seconds, corresponding to more than 36 days.

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.

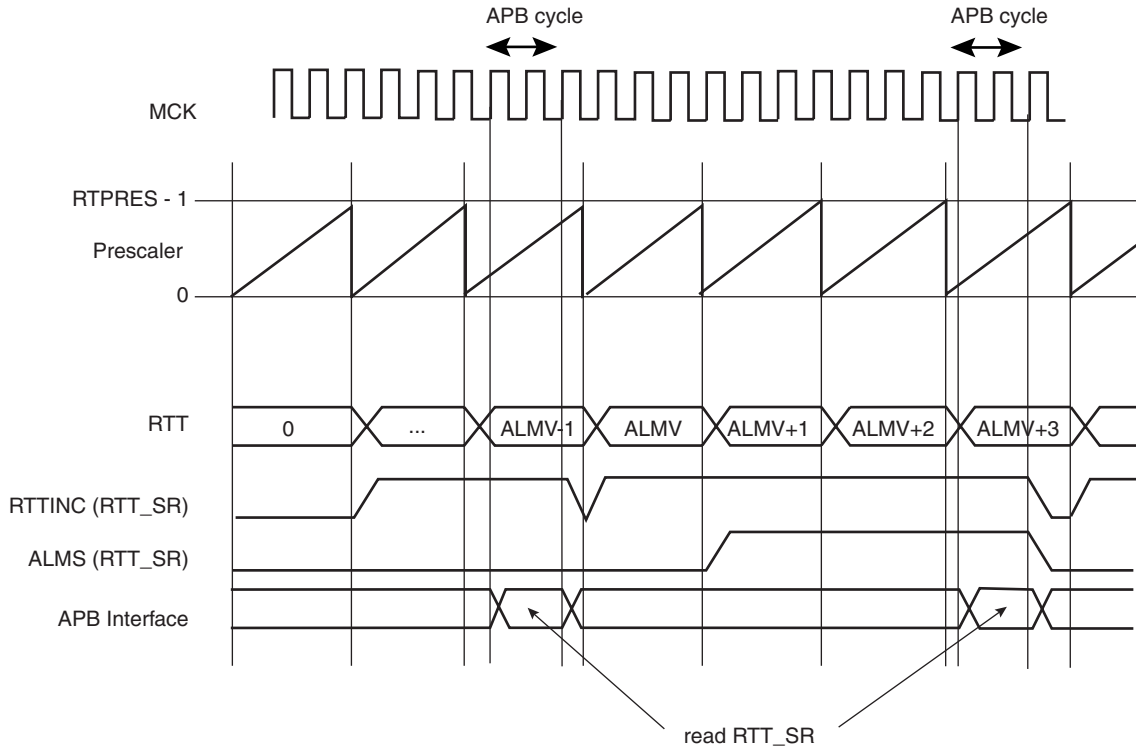
The bit RTTINC in RTT\_SR is set each time the Real-time Timer counter is incremented. This bit can be used to start a periodic interrupt, the period being one second when the RTPRES is programmed with 0x8000 and Slow Clock equal to 32.768 Hz.

Reading the RTT\_SR status register resets the RTTINC and ALMS fields.

Writing the bit RTRRST in RTT\_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.



**Figure 23. RTT Counting**



## Real-time Timer (RTT) User Interface

**Table 12.** Real-time Timer Register Mapping

Offset	Register	Name	Access	Reset Value
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

## 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.

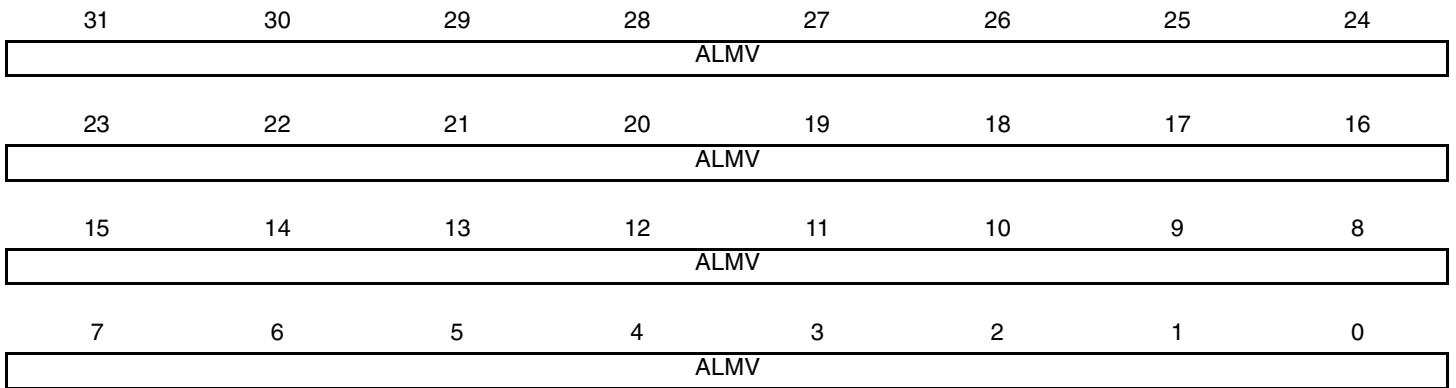
- **RTRRST: Real-time Timer Restart**

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

## Real-time Timer Alarm Register

Register Name: RTT\_AR

Access Type: Read/Write



- **ALMV: Alarm Value**

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

## 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.

## 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.

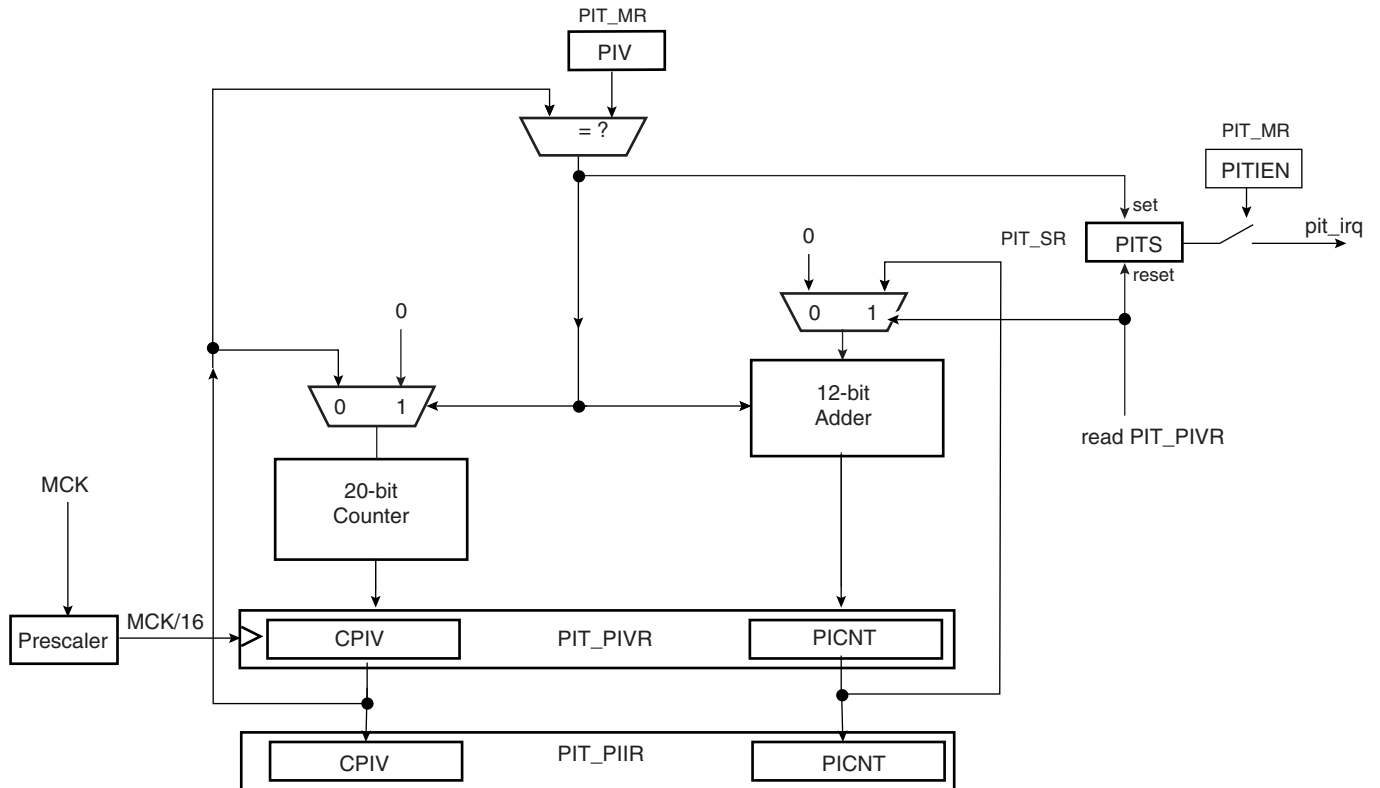
## Periodic Interval Timer (PIT)

### 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.

### Block Diagram

Figure 24. Periodic Interval Timer



## 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.

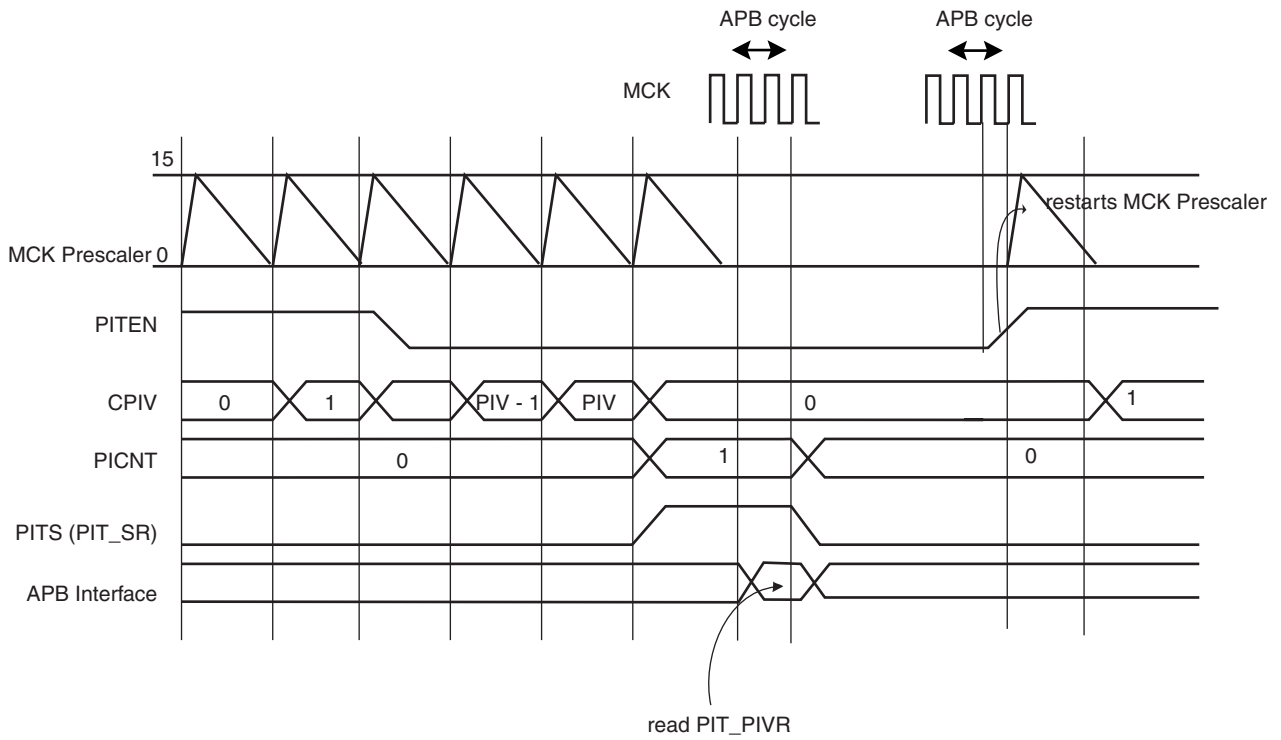
When CPIV and PICNT values are obtained by reading the Periodic Interval Image Register (PIT\_PIIR), there is no effect on the counters CPIV and PICNT, nor on the bit PITS. For example, a profiler can read PIT\_PIIR without clearing any pending interrupt, whereas a timer interrupt clears the interrupt 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 25 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 25. Enabling/Disabling PIT with PITEN**



## Periodic Interval Timer (PIT) User Interface

**Table 13.** 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_PHIR	Read-only	0x0000_0000

## 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.

## 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**

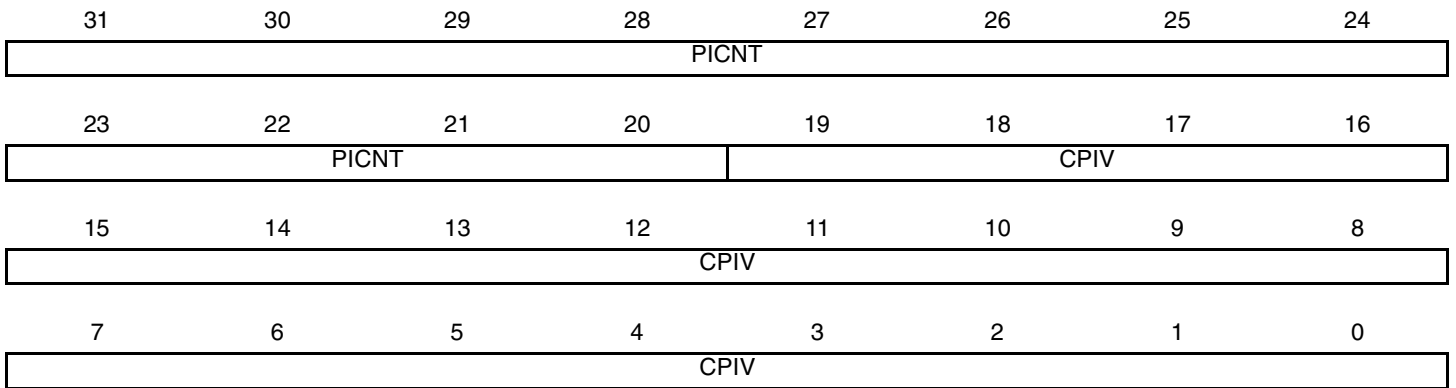
0 = The Periodic Interval timer has not reached PIV since the last read of PIT\_PIVR.

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

## 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**

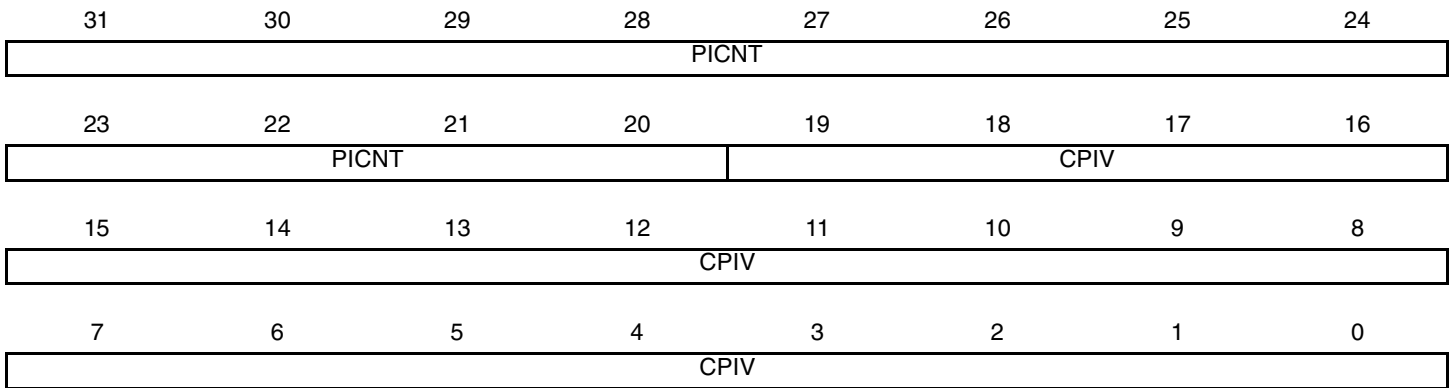
Returns the number of occurrences of periodic intervals since the last read of PIT\_PIVR.



## 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.

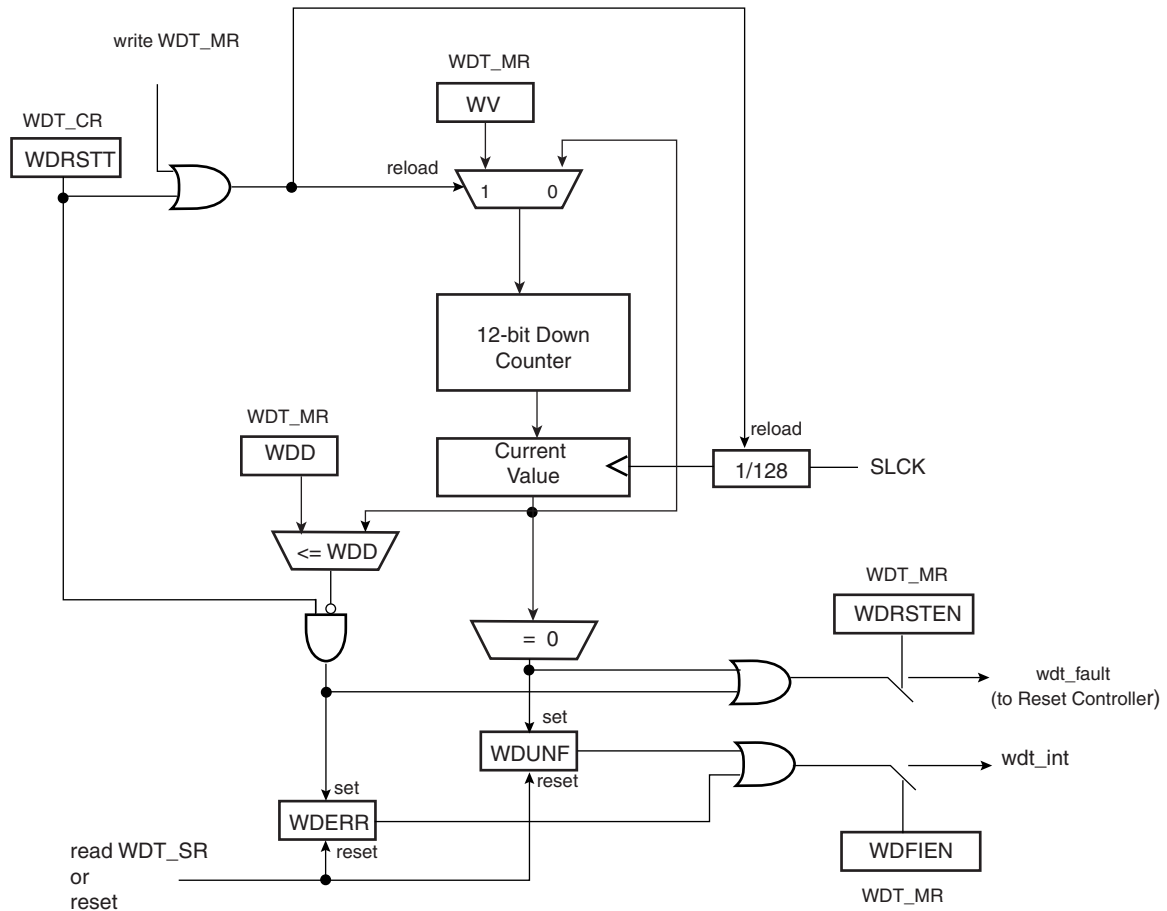
## Watchdog Timer (WDT)

### 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.

### Block Diagram

Figure 26. Watchdog Timer Block Diagram



## 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 WV 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 WV 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 in a window defined by 0 and WDD in the WDT\_MR:

$0 \leq \text{WDT} \leq \text{WDD}$ ; writing WDRSTT restarts the Watchdog Timer.

Any attempt to restart the Watchdog Timer in the range [WDV; 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.

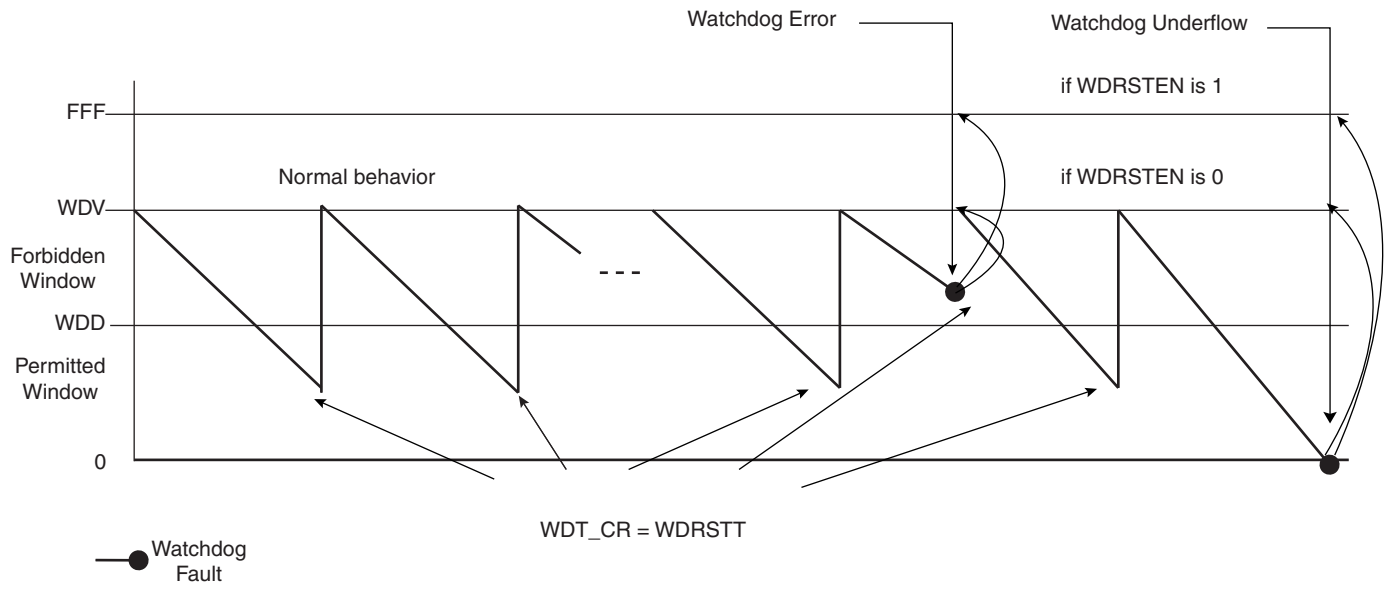
If a reset is generated or if WDT\_SR is read, the status bits are reset, the interrupt is cleared, and the “wdt\_fault” signal to the reset controller is deasserted.

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 27. Watchdog Behavior**



## Watchdog Timer (WDT) User Interface

**Table 14.** 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

## 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.

## 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.

## 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.



## Voltage Regulator Mode Controller (VREG)

### Overview

The Voltage Regulator Mode Controller contains one read/write register, the Voltage Regulator Mode Register. Its offset is 0x60 with respect to the System Controller offset.

This register controls the Voltage Regulator Mode. Setting PSTDBY (bit 0) puts the Voltage Regulator in Standby Mode or Low-power Mode. On reset, the PSTDBY is reset, so as to wake up the Voltage Regulator in Normal Mode.

### Voltage Regulator Power Controller (VREG) User Interface

**Table 15.** Voltage Regulator Power Controller Register Mapping

Offset	Register	Name	Access	Reset Value
0x60	Voltage Regulator Mode Register	VREG_MR	Read/write	0x0

### Voltage Regulator Mode Register

**Register Name:** VREG\_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
–	–	–	–	–	–	–	PSTDBY

- **PSTDBY: Periodic Interval Value**

0 = Voltage regulator in normal mode.

1 = Voltage regulator in standby mode (low-power mode).





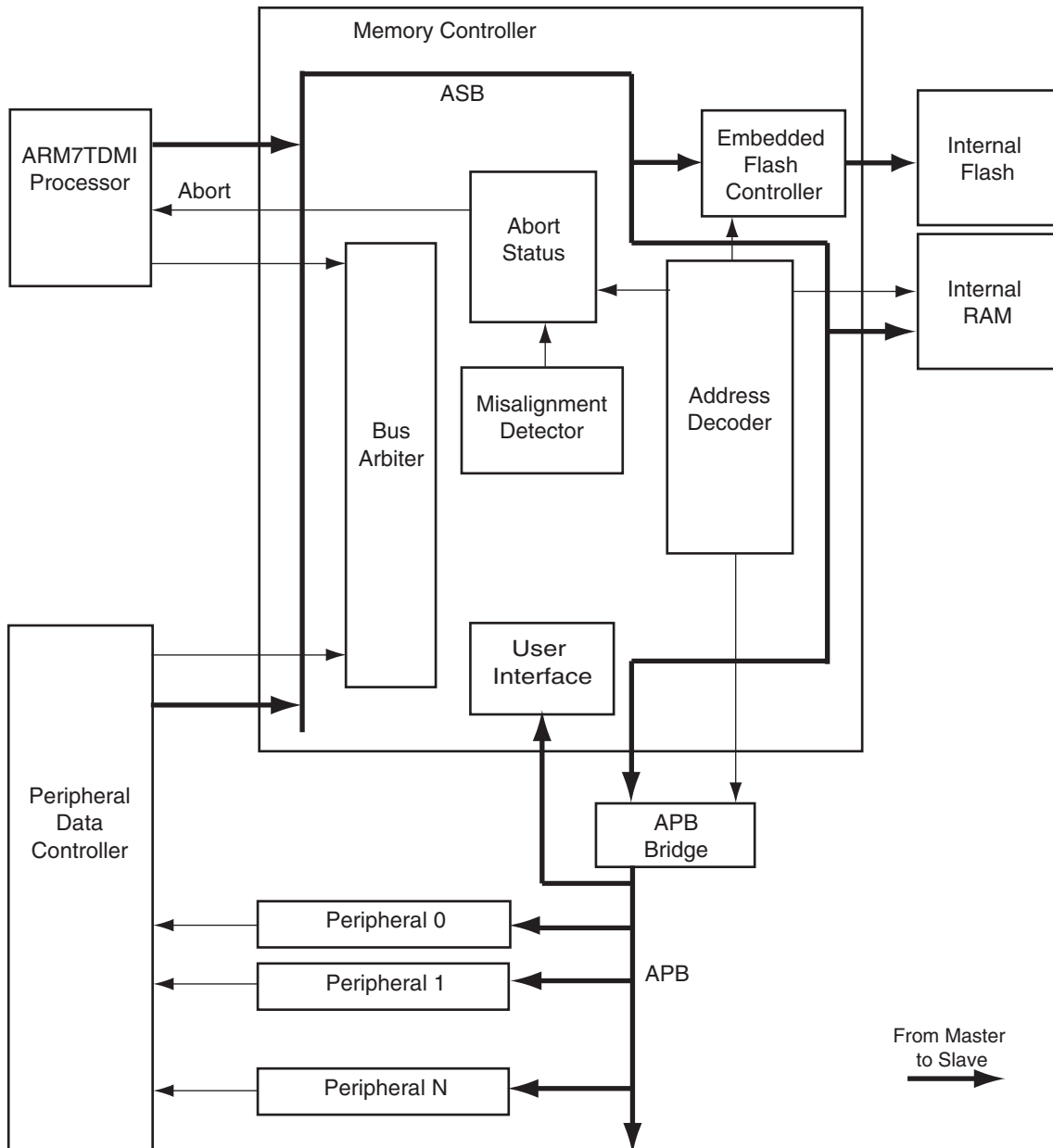
## Memory Controller (MC)

### 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.

### Block Diagram

Figure 28. Memory Controller Block Diagram



## 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
- An Embedded Flash Controller

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

## 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.

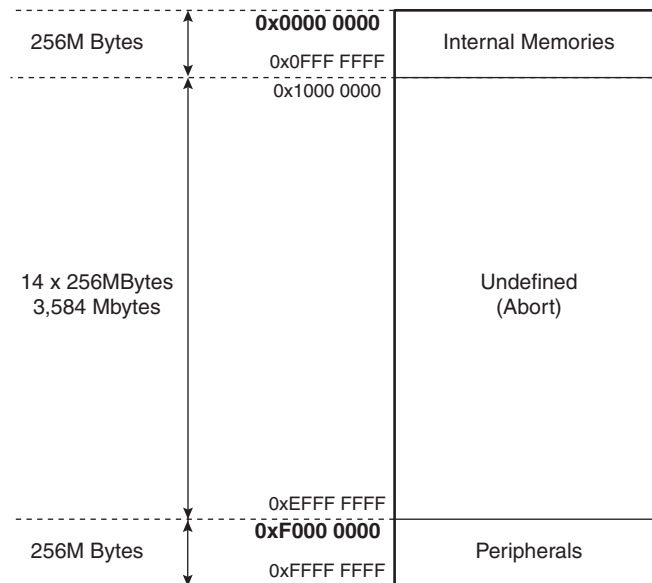
## Address Decoder

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 29 shows the assignment of the 256-Mbyte memory areas.

**Figure 29.** Memory Areas



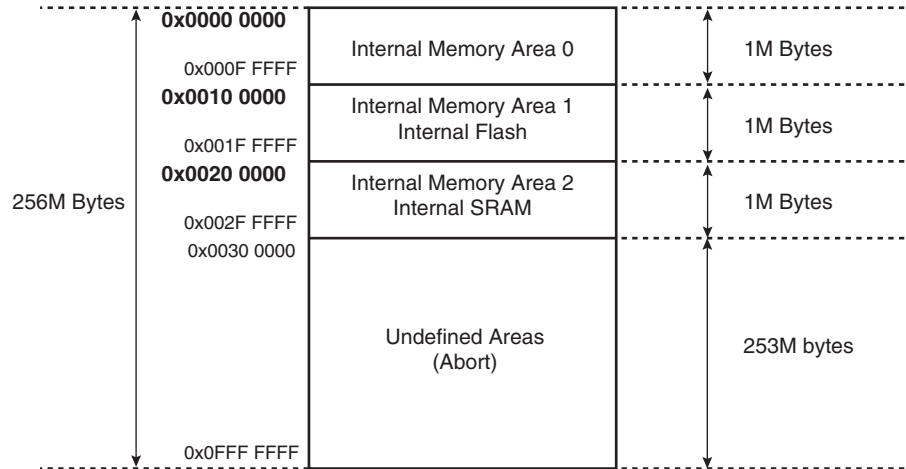
## Internal Memory Mapping

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 30.** Internal Memory Mapping



## 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.

## 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.

## Abort Status

There are three reasons for an abort to occur:

- access to an undefined address
- 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 28.

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) or a misaligned address (bit MISADD)
- 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.

## 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 increases 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.

## 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 bug is simplified.

## Memory Controller (MC) User Interface

Base Address: 0xFFFFF00

**Table 16.** Memory Controller (MC) Register 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-0x5C	Reserved	–	–	–
0x60	EFC Configuration Registers	See “Embedded Flash Controller (EFC)” on page 89.		

## MC Remap Control Register

**Register Name:** MC\_RCR

**Access Type:** Write-only

**Offset:** 0x00

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

## MC Abort Status Register

**Register Name:** MC\_ASR

**Access Type:** Read-only

**Reset Value:** 0x0

**Offset:** 0x04

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
–	–	–	–	–	–	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.

- **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: ARM7TDMI Abort Source**

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

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

- **MST1: PDC Abort Source**

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

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

- **SVMST0: Saved ARM7TDMI Abort Source**

0: No abort due to the ARM7TDMI occurred since the last read of MC\_ASR or it is notified in the bit MST0.

1: At least one abort due to the ARM7TDMI occurred since the last read of MC\_ASR.

- **SVMST1: Saved PDC Abort Source**

0: No abort due to the PDC occurred since the last read of MC\_ASR or it is notified in the bit MST1.

1: At least one abort due to the PDC occurred since the last read of MC\_ASR.

## MC Abort Address Status Register

**Register Name:** MC\_AASR

**Access Type:** Read-only

**Reset Value:** 0x0

**Offset:** 0x08

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.



## Embedded Flash Controller (EFC)

### Overview

The Embedded Flash Controller (EFC) is a part of 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.

### Functional Description

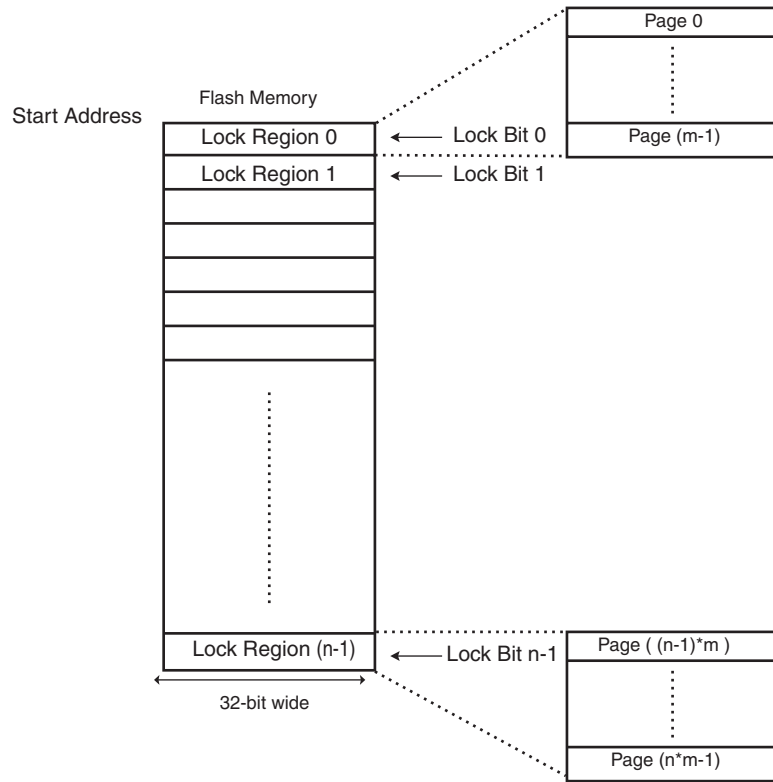
#### Embedded Flash Organization

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 91.)
- 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 93.)
- 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.
- Several general-purpose NVM bits. Each bit controls a specific feature in the device. Refer to the product definition section to get the GP NVM assignment.

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

**Figure 31.** Embedded Flash Memory Mapping



## 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 32, Figure 33 and Figure 34.

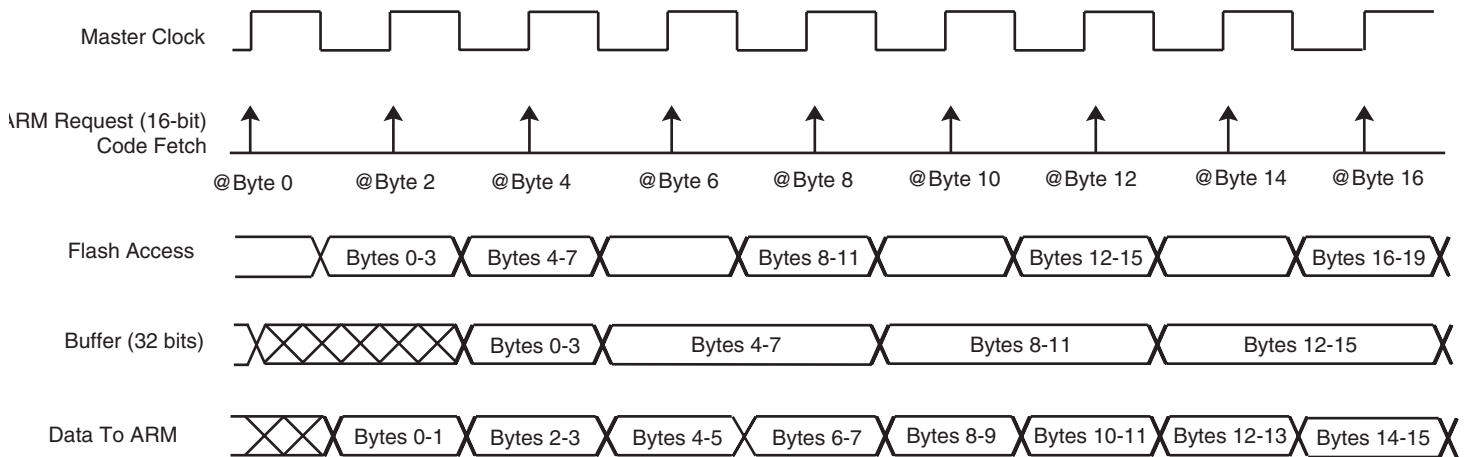
**This optimization concerns only Code Fetch and not Data.**

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 99). 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.

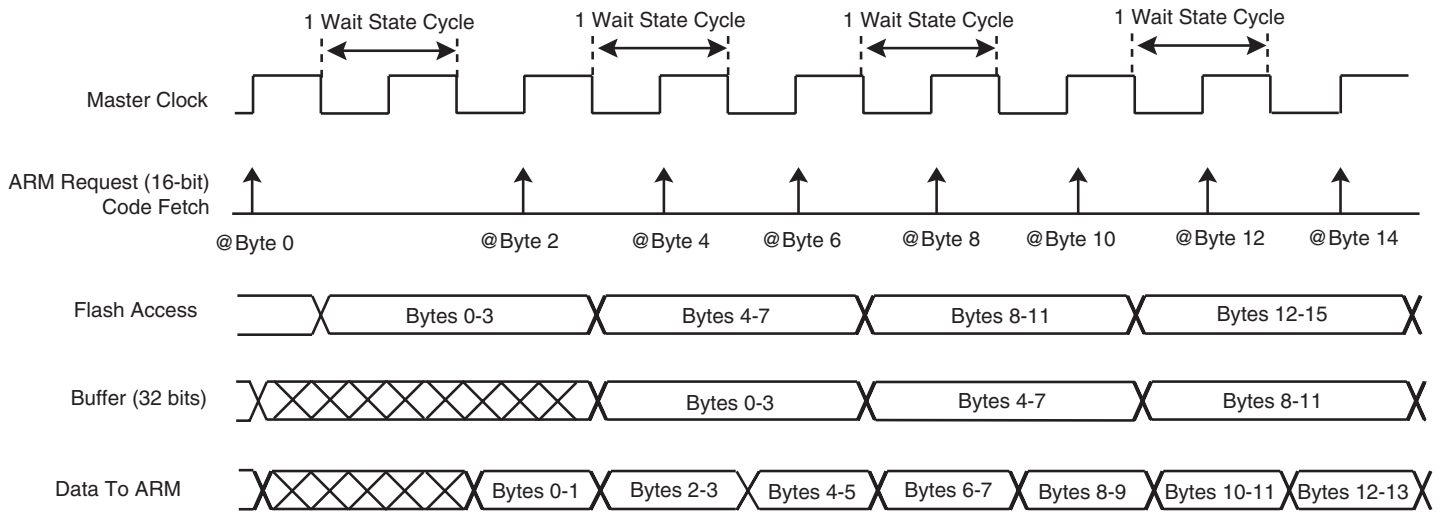
As the Flash block size is smaller than the address space reserved for the internal memory area, the embedded Flash wraps around the address space and appears to be repeated within it.

**Figure 32.** Code Read Optimization in Thumb Mode for FWS = 0



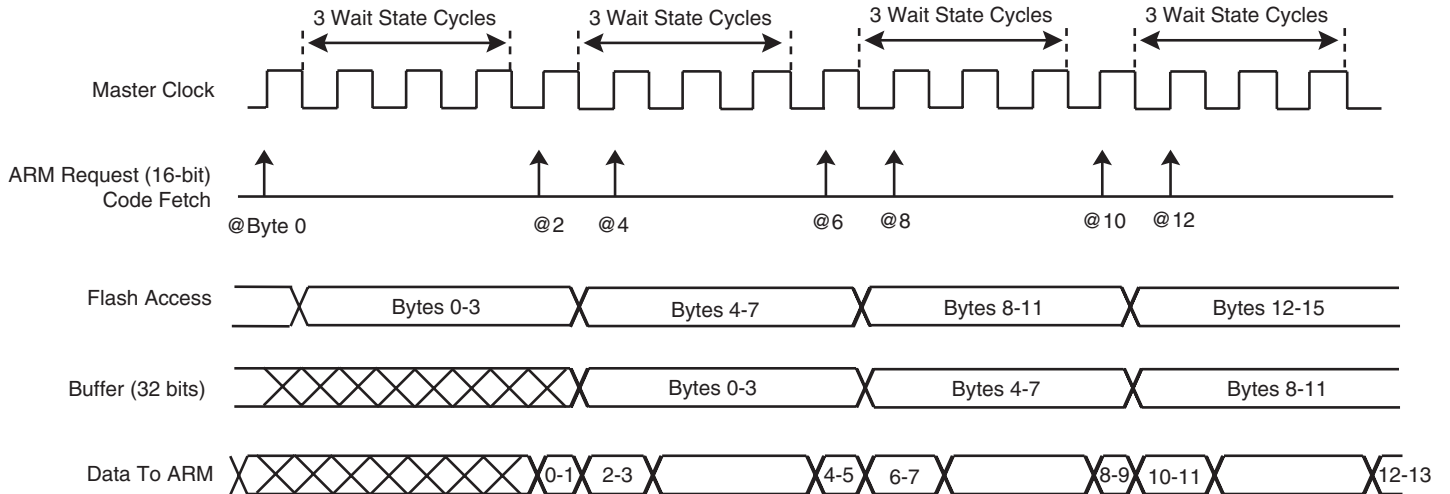
Note: When FWS is equal to 0, all accesses are performed in a single-cycle access.

**Figure 33. 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 34. 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.

## 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 can be prevented by programming the Memory Protection Unit of the product.

**Writing 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 99).

## Flash Commands

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

**Table 17.** 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
Set General-purpose NVM Bit	0x0B	SGPB
Clear General-purpose NVM Bit	0x0D	CGPB
Set Security Bit	0x0F	SSB

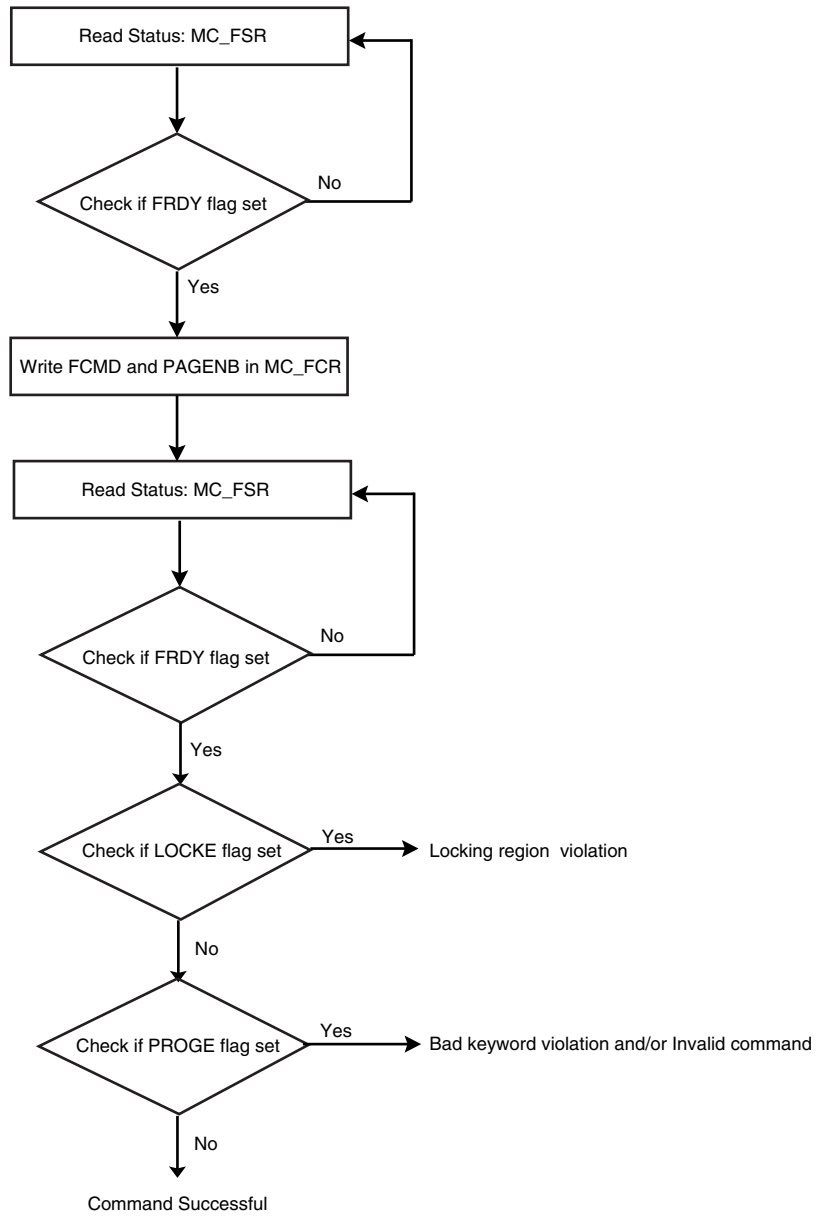
To run one of these commands, the field FCMD of the MC\_FCR register has to be written with the command number. As soon as the MC\_FCR register is written, the FRDY flag is automatically cleared. Once the current command is achieved, then the FRDY flag is automatically set. If an interrupt has been enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

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 FLOCKE flag is set in the MC\_FSR register. This flag is automatically cleared by a read access to the MC\_FSR register.

**Figure 35. Command Statechart**



**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 99).**

Note: This field defines the number of Master Clock cycles in 1 microsecond that allow completion of some necessary internal timings.

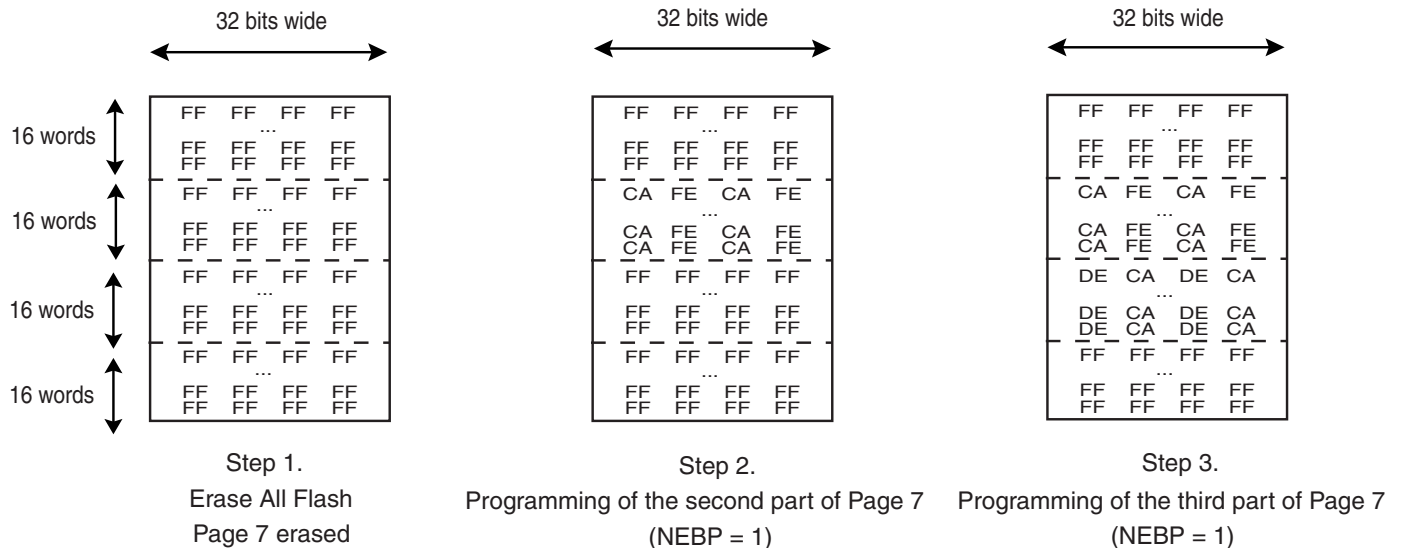
### Flash Programming

Several commands can be used to program the Flash.

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 36).

**Figure 36.** Example of Partial Page Programming:



After programming, the page (the whole lock region) can be locked to prevent miscellaneous write or erase sequences. The lock bit can be automatically set after page programming using WPL.

Data to be written are stored in an internal latch buffer. The size of the latch buffer corresponds to the page size. The latch buffer wraps around within the internal memory area address space and appears to be repeated by the number of pages in it.

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

Data are written to the latch buffer before the programming command is written to 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.
- Programming starts as soon as the page number and the programming command are written to the Flash Command Register. The FRDY bit in the Flash Programming Status Register (MC\_FSR) is automatically cleared.
- When programming is completed, the bit FRDY in the Flash Programming Status Register (MC\_FSR) rises. If an interrupt was enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

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.



## Erase All Command

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.

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

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.

## Lock Bit Protection

Lock bits are associated with several pages in the embedded Flash memory plane. This defines lock regions in the embedded Flash memory plane. They prevent writing/erasing protected pages.

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 sectors can be unlocked to be erased and then programmed with another application or other data.

The lock sequence is:

- The Flash Command register must be written with the following value:  
(0x5A << 24) | (lockPageNumber << 8 & PAGEN) | SLB  
lockPageNumber is a page of the corresponding lock region.
- When locking completes, the bit FRDY in the Flash Programming Status Register (MC\_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

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.

It is possible to clear lock bits that were set previously. Then the locked region can be erased or programmed. The unlock sequence is:

- The Flash Command register must be written with the following value:  
(0x5A << 24) | (lockPageNumber << 8 & PAGEN) | CLB  
lockPageNumber is a page of the corresponding lock region.
- When the unlock completes, the bit FRDY in the Flash Programming Status Register (MC\_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

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.

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

**Note:** Access to the Flash in Read Mode is permitted when a Lock or Unlock command is performed.



## General-purpose NVM Bits

General-purpose NVM bits do not interfere with the embedded Flash memory plane. These general-purpose bits are dedicated to protect other parts of the product. They can be set (activated) or cleared individually. Refer to the product definition section for the general-purpose NVM bit action.

The activation sequence is:

- Start the Set General Purpose Bit command (SGPB) by writing the Flash Command Register with the SEL command and the number of the general-purpose bit to be set in the PAGEN field.
- When the bit is set, the bit FRDY in the Flash Programming Status Register (MC\_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

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
- If the general-purpose bit number is greater than the total number of general-purpose bits, then the command has no effect.

It is possible to deactivate a general-purpose NVM bit set previously. The clear sequence is:

- Start the Clear General-purpose Bit command (CGPB) by writing the Flash Command Register with CGPB and the number of the general-purpose bit to be cleared in the PAGEN field.
- When the clear completes, the bit FRDY in the Flash Programming Status Register (MC\_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

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
- If the number of the general-purpose bit set in the PAGEN field is greater than the total number of general-purpose bits, then the command has no effect.

The Clear General-purpose Bit command programs the general-purpose NVM bit to 1; the corresponding bit GPNVMx in MC\_FSR reads 0. The Set General-purpose Bit command programs the general-purpose NVM bit to 0; the corresponding bit GPNVMx in MC\_FSR reads 1.

**Note:** Access to the Flash in read mode is permitted when a Set, Clear or Get General-purpose NVM Bit command is performed.

## Security Bit

The goal of the security bit is to prevent external access to the internal bus system. JTAG, Fast Flash Programming and Flash Serial Test Interface features are disabled. Once set, this bit can be reset only by an external hardware ERASE request to the chip. Refer to the product definition section for the pin name that controls the ERASE. In this case, the full memory plane is erased and all lock and general-purpose NVM bits are cleared. The security bit in the MC\_FSR is cleared only after these operations.

The activation sequence is:

- Start the Set Security Bit command (SSB) by writing the Flash Command Register.
- When the locking completes, the bit FRDY in the Flash Programming Status Register (MC\_FSR) rises. If an interrupt has been enabled by setting the bit FRDY in MC\_FMR, the interrupt line of the Memory Controller is activated.

When the security bit is active, the SECURITY bit in the MC\_FSR is set.

## Embedded Flash Controller (EFC) User Interface

The User Interface of the Embedded Flash Controller is integrated within the Memory Controller.

**Base Address:** 0xFFFF FF00

**Table 18.** 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	–	–	–

## 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	–	FRDY

- **FRDY: Flash Ready Interrupt Enable**

0 = Flash Ready does not generate an interrupt.  
 1 = Flash Ready 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**

This field defines the number of Master Clock cycles in 1 microsecond.

**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.



## MC Flash Command Register

**Register Name:** MC\_FCR

**Access Type:** Write-only

**Offset:** 0x64

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

- **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	<b>Write Page Command (WP):</b> Starts the programming of the page specified in the PAGEN field.
0010	<b>Set Lock Bit Command (SLB):</b> Starts a set lock bit sequence of the lock region specified in the PAGEN field.
0011	<b>Write Page and Lock Command (WPL):</b> 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	<b>Clear Lock Bit Command (CLB):</b> Starts a clear lock bit sequence of the lock region specified in the PAGEN field.
1000	<b>Erase All Command (EA):</b> Starts the erase of the entire Flash. If at least one page is locked, the command is cancelled.
1011	<b>Set General-purpose NVM Bit (SGPB):</b> Activates the general-purpose NVM bit corresponding to the number specified in the PAGEN field.
1101	<b>Clear General Purpose NVM Bit (CGPB):</b> Deactivates the general-purpose NVM bit corresponding to the number specified in the PAGEN field.
1111	<b>Set Security Bit Command (SSB):</b> Sets security bit.
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.
Set/Clear General Purpose NVM Bit Command	PAGEN defines the general-purpose bit number.
Set Security Bit Command	This field is meaningless

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

- **KEY: Write Protection Key**

This field should be written with the value 0x5A to enable the command defined by the bits of the register. If the field is written with a different value, the write is not performed and no action is started.

## MC Flash Status Register

**Register Name:** MC\_FSR

**Access Type:** Read-only

**Offset:** 0x68

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
LOCKS7	LOCKS6	LOCKS5	LOCKS4	LOCKS3	LOCKS2	LOCKS1	LOCKS0
15	14	13	12	11	10	9	8
–	–	–	–	–	–	GPNVM1	GPNVM0
7	6	5	4	3	2	1	0
–	–	–	SECURITY	PROGE	LOCKE	–	FRDY

- **FRDY: Flash Ready Status**

0 = The Embedded Flash Controller is busy and the application must wait before running a new command.

1 = The Embedded Flash Controller is ready to run a new command.

- **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 keywords were written in the Flash Command Register MC\_FCR.

1 = An invalid command and/or a bad keyword was/were written in the Flash Command Register MC\_FCR.

- **SECURITY: Security Bit Status**

0 = The security bit is disactive.

1 = The security bit is active.

- **GPNVMx: General-purpose NVM Bit Status**

0 = The corresponding general-purpose NVM bit is disactive.

1 = The corresponding general-purpose NVM bit is active.

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

0 = The corresponding lock region is not locked.

1 = The corresponding lock region is locked.

## Fast Flash Programming Interface (FFPI)

### Overview

The Fast Flash Programming Interface provides two solutions - parallel or serial - for high-volume programming using a standard gang programmer. The parallel interface is fully handshaked and the device is considered to be a standard EEPROM. Additionally, the parallel protocol offers an optimized access to all the embedded Flash functionalities. The serial interface uses the standard IEEE 1149.1 JTAG protocol. It offers an optimized access to all the embedded Flash functionalities.

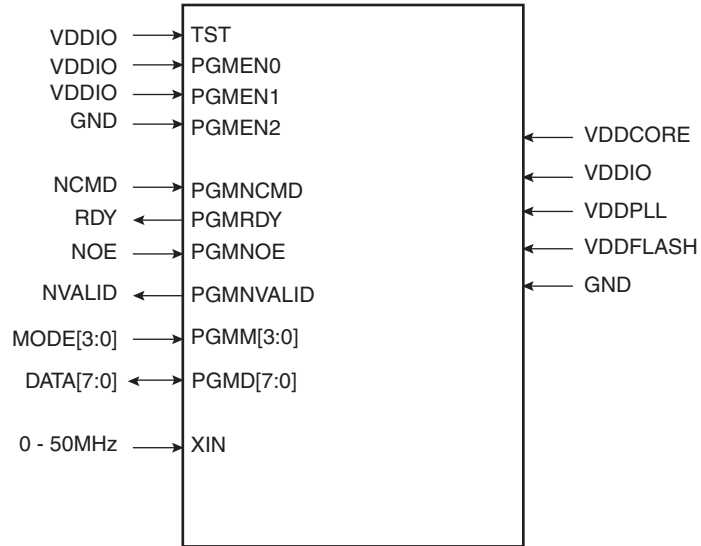
Although the Fast Flash Programming Mode is a dedicated mode for high volume programming, this mode is not designed for in-situ programming.

## Parallel Fast Flash Programming

### Device Configuration

In Fast Flash Programming Mode, the device is in a specific test mode. Only a certain set of pins is significant. Other pins must be left unconnected.

**Figure 37.** Parallel Programming Interface



**Table 19.** Signal Description List

Signal Name	Function	Type	Active Level	Comments
<b>Power</b>				
VDDFLASH	Flash Power Supply	Power		
VDDIO	I/O Lines Power Supply	Power		
VDDCORE	Core Power Supply	Power		
VDDPLL	Backup I/O Lines Power Supply	Power		
GND	Ground	Ground		
<b>Clocks</b>				
XIN	Main Clock Input. This input can be tied to GND. In this case, the device is clocked by the internal RC oscillator.	Input		32KHz to 50MHz
<b>Test</b>				
TST	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN0	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN1	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN2	Test Mode Select	Input	Low	Must be connected to GND
<b>PIO</b>				



**Table 19.** Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Comments
PGMNCMD	Valid command available	Input	Low	Pulled-up input at reset
PGMRDY	0: Device is busy 1: Device is ready for a new command	Output	High	Pulled-up input at reset
PGMNOE	Output Enable (active high)	Input	Low	Pulled-up input at reset
PGMNVALID	0: DATA[7:0] is in input mode 1: DATA[7:0] is in output mode	Output	Low	Pulled-up input at reset
PGMM[3:0]	Specifies DATA type (See Table 20)	Input		Pulled-up input at reset
PGMD[7:0]	Bi-directional data bus	Input/Output		Pulled-up input at reset

## Signal Names

Depending on the MODE settings, DATA is latched in different internal registers.

**Table 20.** Mode Coding

MODE[3:0]	Symbol	Data
0000	CMDE	Command Register
0001	ADDR0	Address Register LSBs
0010	ADDR1	
0011	ADDR2	
0100	ADDR2	Address Register MSBs
0101	DATA	Data Register
Default	IDLE	No register

When MODE is equal to CMDE, then a new command (strobed on DATA[7:0] signals) is stored in the command register.

**Table 21.** Command Bit Coding

DATA[7:0]	Symbol	Command Executed
0x0011	READ	Read Flash
0x0012	WP	Write Page Flash
0x0022	WPL	Write Page and Lock Flash
0x0032	EWP	Erase Page and Write Page
0x0042	EWPL	Erase Page and Write Page then Lock
0x0013	EA	Erase All
0x0014	SLB	Set Lock Bit
0x0024	CLB	Clear Lock Bit
0x0015	GLB	Get Lock Bit
0x0034	SFB	Set General Purpose NVM bit
0x0044	CFB	Clear General Purpose NVM bit
0x0025	GFB	Get General Purpose NVM bit

**Table 21.** Command Bit Coding (Continued)

DATA[7:0]	Symbol	Command Executed
0x0054	SSE	Set Security Bit
0x0035	GSE	Get Security Bit
0x001E	GVE	Get Version

## Entering Programming Mode

The following algorithm puts the device in Parallel Programming Mode:

- Apply GND, VDDIO, VDDCORE, VDDFLASH and VDDPLL.
- Apply XIN clock within  $T_{POR\_RESET}$  if an external clock is available.
- Wait for  $T_{POR\_RESET}$
- Start a read or write handshaking.

Note: After reset, the device is clocked by the internal RC oscillator. Before clearing RDY signal, if an external clock (> 32 kHz) is connected to XIN, then the device switches on the external clock. Else, XIN input is not considered. A higher frequency on XIN speeds up the programmer handshake.

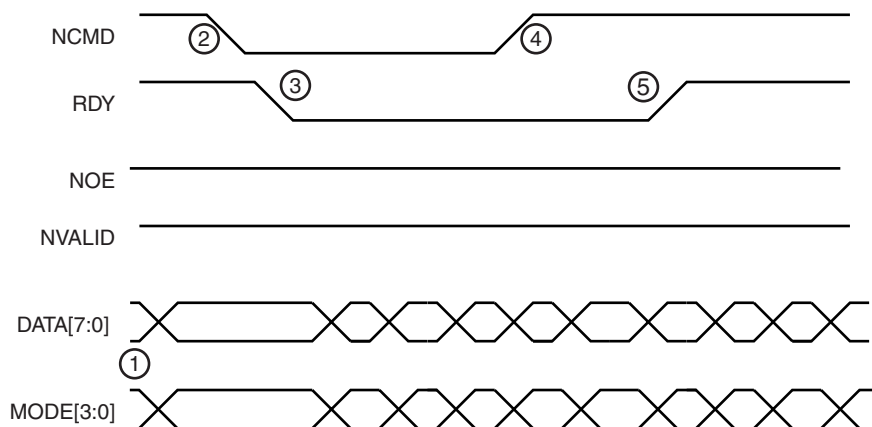
## Programmer Handshaking

An handshake is defined for read and write operations. When the device is ready to start a new operation (RDY signal set), the programmer starts the handshake by clearing the NCMD signal. The handshaking is achieved once NCMD signal is high and RDY is high.

## Write Handshaking

For details on the write handshaking sequence, refer to Figure 38 and Table 22.

**Figure 38.** Parallel Programming Timing, Write Sequence



**Table 22.** Write Handshake

Step	Programmer Action	Device Action	Data I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latches MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input

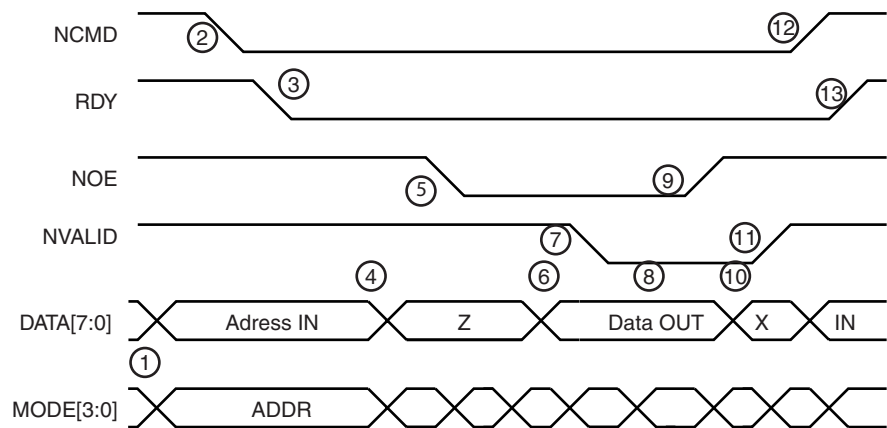
**Table 22.** Write Handshake (Continued)

Step	Programmer Action	Device Action	Data I/O
4	Releases MODE and DATA signals	Executes command and polls NCMD high	Input
5	Sets NCMD signal	Executes command and polls NCMD high	Input
6	Waits for RDY high	Sets RDY	Input

## Read Handshaking

For details on the read handshaking sequence, refer to Figure 39 and Table 23.

**Figure 39.** Parallel Programming Timing, Read Sequence



**Table 23.** Read Handshake

Step	Programmer Action	Device Action	DATA I/O
1	Sets MODE and DATA signals	Waits for NCMD low	Input
2	Clears NCMD signal	Latch MODE and DATA	Input
3	Waits for RDY low	Clears RDY signal	Input
4	Sets DATA signal in tristate	Waits for NOE Low	Input
5	Clears NOE signal		Tristate
6	Waits for NVALID low	Sets DATA bus in output mode and outputs the flash contents.	Output
7		Clears NVALID signal	Output
8	Reads value on DATA Bus	Waits for NOE high	Output
9	Sets NOE signal		Output
10	Waits for NVALID high	Sets DATA bus in input mode	X
11	Sets DATA in output mode	Sets NVALID signal	Input
12	Sets NCMD signal	Waits for NCMD high	Input
13	Waits for RDY high	Sets RDY signal	Input

## Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 21 on page 105. Each command is driven by the programmer through the parallel interface running several read/write handshaking sequences.

When a new command is executed, the previous one is automatically achieved. Thus, chaining a read command after a write automatically flushes the load buffer in the Flash.

### Flash Read Command

This command is used to read the contents of the Flash memory. The read command can start at any valid address in the memory plane and is optimized for consecutive reads. Read handshaking can be chained; an internal address buffer is automatically increased.

**Table 24.** Read Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	READ
2	Write handshaking	ADDR0	32-bit Memory Address First byte
3	Write handshaking	ADDR1	32-bit Flash Address
4	Write handshaking	ADDR2	32-bit Flash Address
5	Write handshaking	ADDR3	32-bit Flash Address Last Byte
6	Read handshaking	DATA	*Memory Address++
7	Read handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	32-bit Memory Address First byte
n+1	Write handshaking	ADDR1	32-bit Flash Address
n+2	Write handshaking	ADDR2	32-bit Flash Address
n+3	Write handshaking	ADDR3	32-bit Flash Address Last Byte
n+4	Read handshaking	DATA	*Memory Address++
n+5	Read handshaking	DATA	*Memory Address++
...	...	...	...

### Flash Write Command

This command is used to write the Flash contents.

The Flash memory plane is organized into several pages. Data to be written are stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- before access to any page other than the current one
- when a new command is validated (MODE = CMDE)

The **Write Page** command (**WP**) is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

**Table 25.** Write Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	WP or WPL or EWP or EWPL
2	Write handshaking	ADDR0	32-bit Memory Address First byte
3	Write handshaking	ADDR1	32-bit Flash Address
4	Write handshaking	ADDR2	32-bit Flash Address

**Table 25.** Write Command (Continued)

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
5	Write handshaking	ADDR3	32-bit Flash Address Last Byte
6	Write handshaking	DATA	*Memory Address++
7	Write handshaking	DATA	*Memory Address++
...	...	...	...
n	Write handshaking	ADDR0	32-bit Memory Address First byte
n+1	Write handshaking	ADDR1	32-bit Flash Address
n+2	Write handshaking	ADDR2	32-bit Flash Address
n+3	Write handshaking	ADDR3	32-bit Flash Address Last Byte
n+4	Write handshaking	DATA	*Memory Address++
n+5	Write handshaking	DATA	*Memory Address++
...	...	...	...

The Flash command **Write Page and Lock (WPL)** is equivalent to the Flash Write Command. However, the lock bit is automatically set at the end of the Flash write operation. As a lock region is composed of several pages, the programmer writes to the first pages of the lock region using Flash write commands and writes to the last page of the lock region using a Flash write and lock command.

The Flash command **Erase Page and Write (EWP)** is equivalent to the Flash Write Command. However, before programming the load buffer, the page is erased.

The Flash command **Erase Page and Write the Lock (EWPL)** combines EWP and WPL commands.

## Flash Full Erase Command

This command is used to erase the Flash memory planes.

All lock regions must be unlocked before the Full Erase command by using the CLB command. Otherwise, the erase command is aborted and no page is erased.

**Table 26.** Full Erase Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	EA
2	Write handshaking	DATA	0

## Flash Lock Commands

Lock bits can be set using WPL or EWPL commands. They can also be set by using the **Set Lock** command (**SLB**). With this command, several lock bits can be activated. A Bit Mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first lock bit is activated.

In the same way, the **Clear Lock** command (**CLB**) is used to clear lock bits. All the lock bits are also cleared by the EA command.

**Table 27.** Set and Clear Lock Bit Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	SLB or CLB
2	Write handshaking	DATA	Bit Mask

Lock bits can be read using **Get Lock Bit** command (**GLB**). The  $n^{\text{th}}$  lock bit is active when the bit  $n$  of the bit mask is set..

**Table 28.** Get Lock Bit Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	GLB
2	Read handshaking	DATA	Lock Bit Mask Status 0 = Lock bit is cleared 1 = Lock bit is set

## Flash General Purpose NVM Commands

General-purpose NVM bits (GP NVM bits) can be set using the **Set Fuse** command (**SFB**). This command also activates GP NVM bits. A bit mask is provided as argument to the command. When bit 0 of the bit mask is set, then the first GP NVM bit is activated.

In the same way, the **Clear Fuse** command (**CFB**) is used to clear general-purpose NVM bits. All the general-purpose NVM bits are also cleared by the EA command. The general-purpose NVM bit is deactivated when the corresponding bit in the pattern value is set to 1.

**Table 29.** Set/Clear GP NVM Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	SFB or CFB
2	Write handshaking	DATA	GP NVM bit pattern value

General-purpose NVM bits can be read using the **Get Fuse Bit** command (**GFB**). The  $n^{\text{th}}$  GP NVM bit is active when bit  $n$  of the bit mask is set..

**Table 30.** Get GP NVM Bit Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	GFB
2	Read handshaking	DATA	GP NVM Bit Mask Status 0 = GP NVM bit is cleared 1 = GP NVM bit is set

## Flash Security Bit Command

A security bit can be set using the **Set Security Bit** command (SSE). Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. An event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

**Table 31.** Set Security Bit Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	SSE
2	Write handshaking	DATA	0

## Get Version Command

The **Get Version** (GVE) command retrieves the version of the FFPI interface.

**Table 32.** Get Version Command

Step	Handshake Sequence	MODE[3:0]	DATA[7:0]
1	Write handshaking	CMDE	GVE
2	Write handshaking	DATA	Version

## Serial Fast Flash Programming

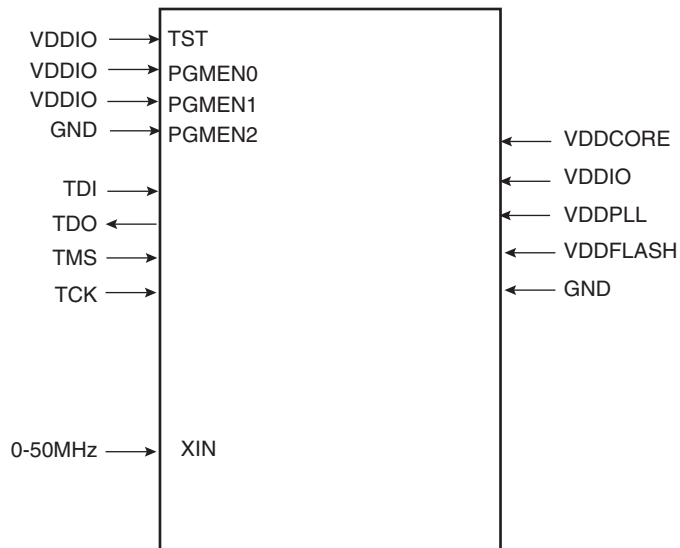
The Serial Fast Flash programming interface is based on IEEE Std. 1149.1 “Standard Test Access Port and Boundary-Scan Architecture”. Refer to this standard for an explanation of terms used in this chapter and for a description of the TAP controller states.

In this mode, data read/written from/to the embedded Flash of the device are transmitted through the JTAG interface of the device.

### Device Configuration

In Serial Fast Flash Programming Mode, the device is in a specific test mode. Only a distinct set of pins is significant. Other pins must be left unconnected.

**Figure 40.** Serial Programming



**Table 33.** Signal Description List

Signal Name	Function	Type	Active Level	Comments
<b>Power</b>				
VDDFLASH	Flash Power Supply	Power		
VDDIO	I/O Lines Power Supply	Power		
VDDCORE	Core Power Supply	Power		
VDDPLL	Backup I/O Lines Power Supply	Power		
GND	Ground	Ground		
<b>Clocks</b>				
XIN	Main Clock Input. This input can be tied to GND. In this case, the device is clocked by the internal RC oscillator.	Input		32 kHz to 50 MHz



**Table 33.** Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Comments
<b>Test</b>				
TST	Test Mode Select	Input	High	Must be connected to VDDIO.
PGMEN0	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN1	Test Mode Select	Input	High	Must be connected to VDDIO
PGMEN2	Test Mode Select	Input	Low	Must be connected to GND
<b>JTAG</b>				
TCK	JTAG TCK	Input	-	Pulled-up input at reset
TDI	JTAG Test Data In	Input	-	Pulled-up input at reset
TDO	JTAG Test Data Out	Output	-	
TMS	JTAG Test Mode Select	Input	-	Pulled-up input at reset

## Entering Serial Programming Mode

The following algorithm puts the device in Serial Programming Mode:

- Apply GND, VDDIO, VDDCORE, VDDFLASH and VDDPLL.
- Apply XIN clock within  $T_{POR\_RESET} + 32(T_{SCLK})$  if an external clock is available.
- Wait for  $T_{POR\_RESET}$ .
- Reset the TAP controller clocking 5 TCK pulses with TMS set.
- Shift 0x2 into the IR register ( IR is 4 bits long, LSB first) without going through the Run-Test-Idle state.
- Shift 0x2 into the DR register ( DR is 4 bits long, LSB first) without going through the Run-Test-Idle state.
- Shift 0xC into the IR register ( IR is 4 bits long, LSB first) without going through the Run-Test-Idle state.

Note: After reset, the device is clocked by the internal RC oscillator. Before clearing RDY signal, if an external clock (> 32 kHz) is connected to XIN, then the device will switch on the external clock. Else, XIN input is not considered. An higher frequency on XIN speeds up the programmer handshake.

**Table 34.** Reset TAP controller and go to Select-DR-Scan

TDI	TMS	TAP Controller State
X	1	
X	1	
X	1	
X	1	
X	1	Test-Logic Reset
X	0	Run-Test/Idle
Xt	1	Select-DR-Scan

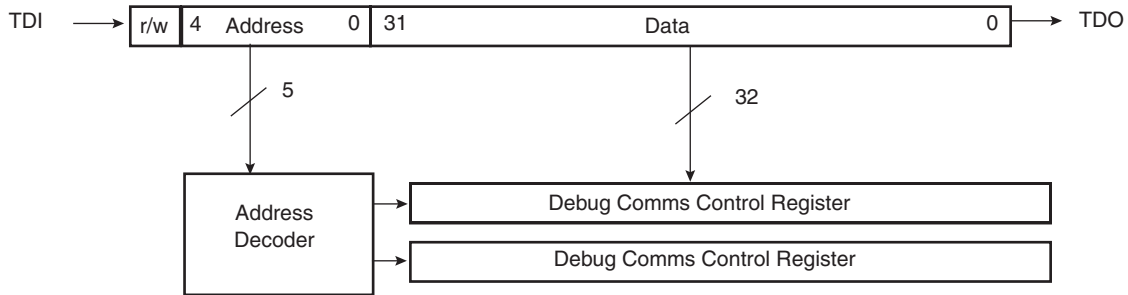
## Read/Write Handshake

Two registers of the device are accessible through the JTAG:

- Debug Comms Control Register: DCCR
- Debug Comms Data Register: DCDR

Access to these registers is done through the TAP 38-bit DR register comprising a 32-bit data field, a 5-bit address field and a read/write bit. The data to be written is scanned into the 32-bit data field with the address of the register to the 5-bit address field and 1 to the read/write bit. A register is read by scanning its address into the address field and 0 into the read/write bit, going through the UPDATE-DR TAP state, then scanning out the data. The 32-bit data field is ignored.

**Figure 41.** TAP 8-bit DR Register



A read or write takes place when the TAP controller enters UPDATE-DR state.

- The address of the Debug Comms Control Register is 0x04.
- The address of the Debug Comms Data Register is 0x05.

The Debug Comms Control Register is read-only and allows synchronized handshaking between the processor and the debugger.

- Bit 1 (W): Denotes whether the programmer can read a data through the Debug Comms Data Register. If the device is busy  $W = 0$ , then the programmer must poll until  $W = 1$ .
- Bit 0 (R): Denotes whether the programmer can send data from the Debug Comms Data Register. If  $R = 1$ , data previously placed there through the scan chain has not been collected by the device and so the programmer must wait.

## Device Operations

Several commands on the Flash memory are available. These commands are summarized in Table 21 on page 105. Commands are run by the programmer through the serial interface that is reading and writing the Debug Comms Registers.

### Flash Read Command

This command is used to read the Flash contents. The memory map is accessible through this command. Memory is seen as an array of words (32-bit wide). The read command can start at any valid address in the memory plane. **This address must be word-aligned.** The address is automatically incremented.

**Table 35.** Read Command

Read/Write	DR Data
Write	(Number of Words to Read) << 16   READ
Write	Address
Read	Memory [address]
Read	Memory [address+4]
...	...
Read	Memory [address+(Number of Words to Read - 1)* 4]

## Flash Write Command

This command is used to write the Flash contents. The address transmitted must be a valid Flash address in the memory plane.

The Flash memory plane is organized into several pages. Data to be written is stored in a load buffer that corresponds to a Flash memory page. The load buffer is automatically flushed to the Flash:

- before access to any page than the current one
- at the end of the number of words transmitted

The **Write Page** command (**WP**) is optimized for consecutive writes. Write handshaking can be chained; an internal address buffer is automatically increased.

**Table 36.** Write Command

Read/Write	DR Data
Write	(Number of Words to Write) << 16   (WP or WPL or EWP or EWPL)
Write	Address
Write	Memory [address]
Write	Memory [address+4]
Write	Memory [address+8]
Write	Memory [address+(Number of Words to Write - 1)* 4]

Flash **Write Page and Lock** command (**WPL**) is equivalent to the Flash Write Command. However, the lock bit is automatically set at the end of the Flash write operation. As a lock region is composed of several pages, the programmer writes to the first pages of the lock region using Flash write commands and writes to the last page of the lock region using a Flash write and lock command.

Flash **Erase Page and Write** command (**EWP**) is equivalent to the Flash Write Command. However, before programming the load buffer, the page is erased.

Flash **Erase Page and Write the Lock** command (**EWPL**) combines EWP and WPL commands.

## Flash Full Erase Command

This command is used to erase the Flash memory planes.

All lock bits must be deactivated before using the **Full Erase** command. This can be done by using the CLB command.

**Table 37.** Full Erase Command

Read/Write	DR Data
Write	EA

## Flash Lock Commands

Lock bits can be set using WPL or EWPL commands. They can also be set by using the **Set Lock** command (**SLB**). With this command, several lock bits can be activated at the same time. Bit 0 of Bit Mask corresponds to the first lock bit and so on.

In the same way, the **Clear Lock** command (**CLB**) is used to clear lock bits. All the lock bits can also be cleared by the EA command.

**Table 38.** Set and Clear Lock Bit Command

Read/Write	DR Data
Write	SLB or CLB
Write	Bit Mask

Lock bits can be read using **Get Lock Bit** command (**GLB**). When a bit set in the Bit Mask is returned, then the corresponding lock bit is active.

**Table 39.** Get Lock Bit Command

Read/Write	DR Data
Write	GLB
Read	Bit Mask

### Flash General-purpose NVM Commands

General-purpose NVM bits (GP NVM) can be set with the **Set Fuse** command (**SFB**). Using this command, several GP NVM bits can be activated at the same time. Bit 0 of Bit Mask corresponds to the first fuse bit and so on.

In the same way, the **Clear Fuse** command (**CFB**) is used to clear GP NVM bits. All the general-purpose NVM bits are also cleared by the EA command.

**Table 40.** Set and Clear General-purpose NVM Bit Command

Read/Write	DR Data
Write	SFB or CFB
Write	Bit Mask

GP NVM bits can be read using **Get Fuse Bit** command (**GFB**). When a bit set in the Bit Mask is returned, then the corresponding fuse bit is set.

**Table 41.** Get General-purpose NVM Bit Command

Read/Write	DR Data
Write	GFB
Read	Bit Mask

### Flash Security Bit Command

Security bits can be set using **Set Security Bit** command (SSE). Once the security bit is active, the Fast Flash programming is disabled. No other command can be run. Only an event on the Erase pin can erase the security bit once the contents of the Flash have been erased.

**Table 42.** Set Security Bit Command

Read/Write	DR Data
Write	SSE

### Get Version Command

The **Get Version** (GVE) command retrieves the version of the FFPI interface.

**Table 43.** Get Version Command

Read/Write	DR Data
Write	GVE
Read	Version

## Peripheral Data Controller (PDC)

### Overview

The Peripheral Data 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 Data 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.

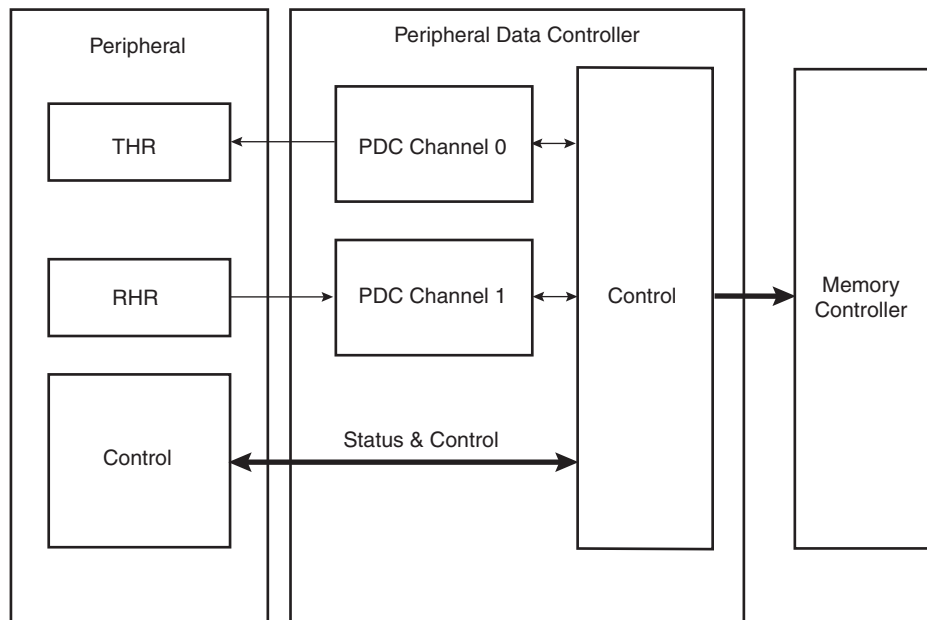
The user interface of a PDC channel is integrated in the memory space of each peripheral. It contains:

- A 32-bit memory pointer register
- A 16-bit transfer count register
- A 32-bit register for next memory pointer
- A 16-bit register for next transfer count

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.

### Block Diagram

Figure 42. Block Diagram



## Functional Description

### 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.

### 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.

### 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, ENTX) 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.

## 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.

## Priority of PDC Transfer Requests

The Peripheral Data 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.

## Peripheral Data Controller (PDC) User Interface

**Table 44.** Peripheral Data Controller (PDC) 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).



## 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.

## 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.

## 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.

## 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.

## 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.

## 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.



## 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.

## 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.

## 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

## 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.

## Advanced Interrupt Controller (AIC)

### 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.

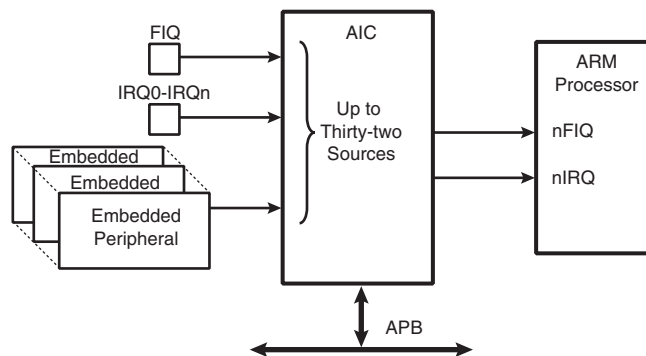
The 8-level Priority Controller allows the user to define the priority for each interrupt source, thus permitting higher priority interrupts to be serviced even if a lower priority interrupt is being treated.

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.

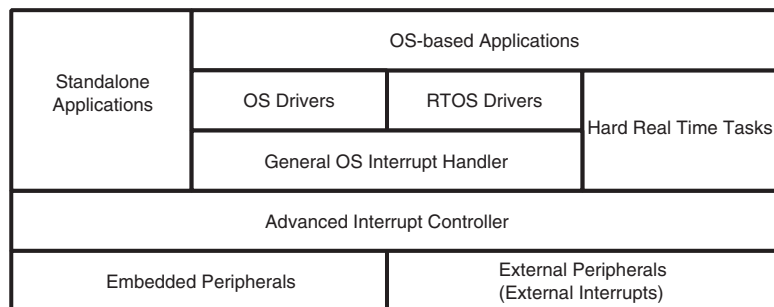
### Block Diagram

Figure 43. Block Diagram



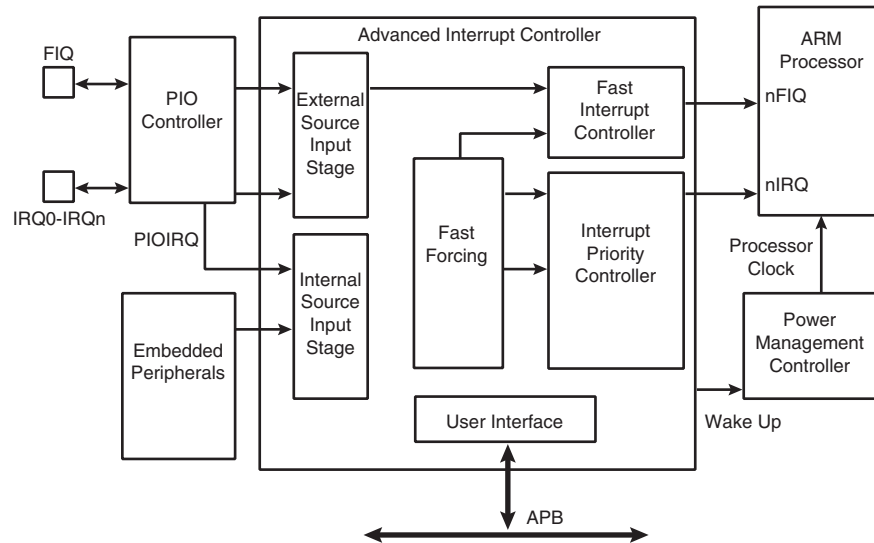
### Application Block Diagram

Figure 44. Description of the Application Block



## AIC Detailed Block Diagram

Figure 45. AIC Detailed Block Diagram



## I/O Line Description

Table 45. I/O Line Description

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

## Product Dependencies

### 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.

### 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.

### 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.



## Functional Description

### Interrupt Source Control

**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.

### 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.

### 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 133.) 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 137.)

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

### Interrupt Status

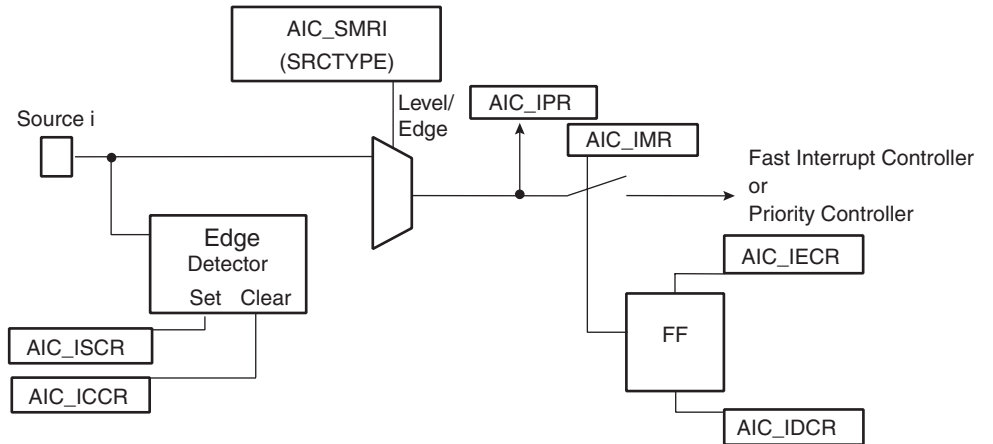
For each interrupt, the AIC operation originates in AIC\_IPR (Interrupt Pending Register) and its mask in AIC\_IMR (Interrupt Mask Register). AIC\_IPR enables the actual activity of the sources, whether masked or not.

The AIC\_ISR register reads the number of the current interrupt (see “Priority Controller” on page 133) 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.

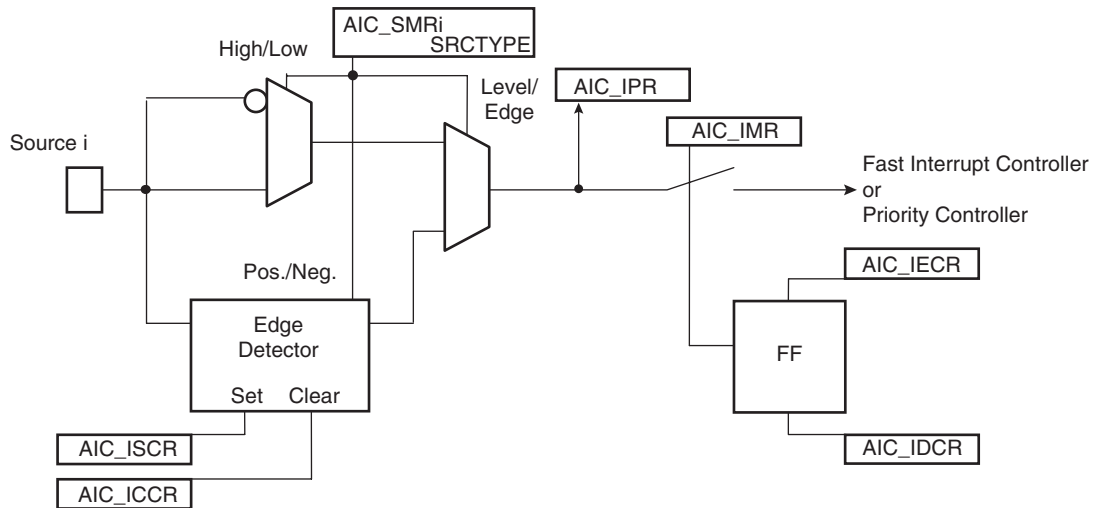
## Internal Interrupt Source Input Stage

**Figure 46.** Internal Interrupt Source Input Stage



## External Interrupt Source Input Stage

**Figure 47.** External Interrupt Source Input Stage



## Interrupt Latencies

Global interrupt latencies depend on several parameters, including:

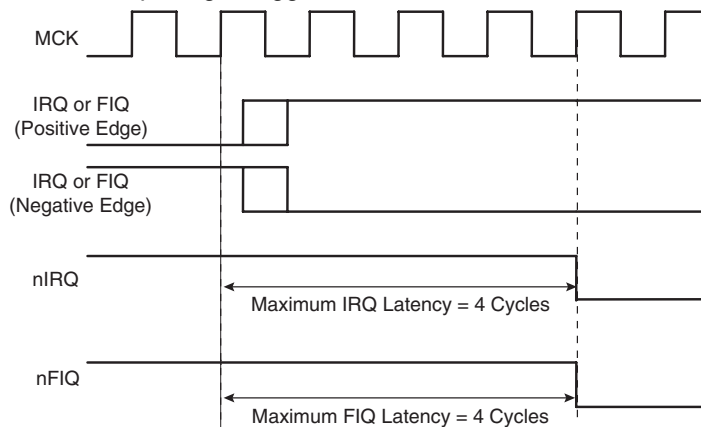
- The time the software masks the interrupts.
- Occurrence, either at the processor level or at the AIC level.
- The execution time of the instruction in progress when the interrupt occurs.
- The treatment of higher priority interrupts and the resynchronization of the hardware signals.

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.

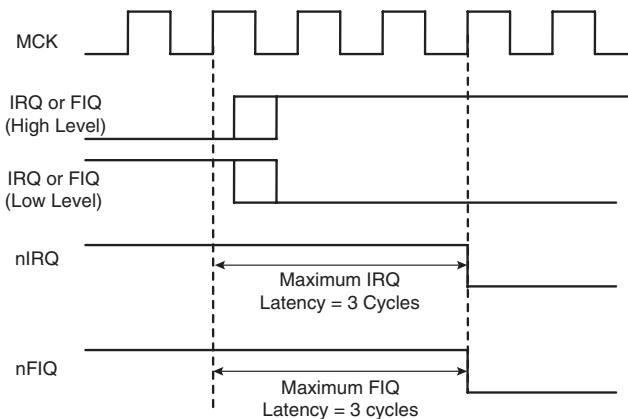
### External Interrupt Edge Triggered Source

**Figure 48.** External Interrupt Edge Triggered Source



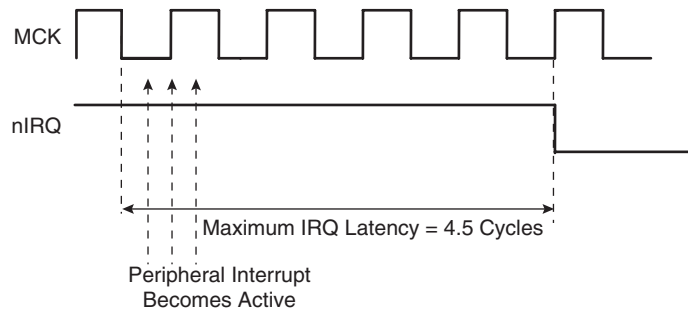
### External Interrupt Level Sensitive Source

**Figure 49.** External Interrupt Level Sensitive Source



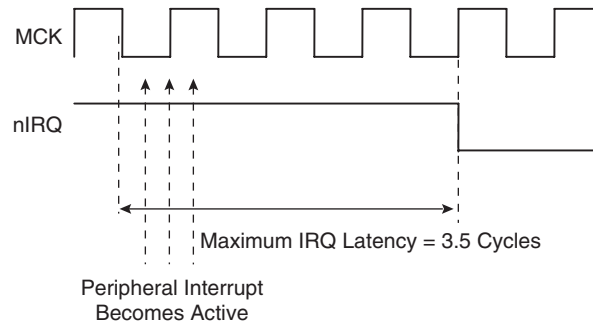
## Internal Interrupt Edge Triggered Source

**Figure 50.** Internal Interrupt Edge Triggered Source



## Internal Interrupt Level Sensitive Source

**Figure 51.** Internal Interrupt Level Sensitive Source



## Normal Interrupt

### 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\_SVR (Source Vector 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.**

## 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.

## 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.

## 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).

## Fast Interrupt

### 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.

### 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\_IEMR (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.

### 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.

### 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 automatically 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.

## 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\_FFSSR) 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).

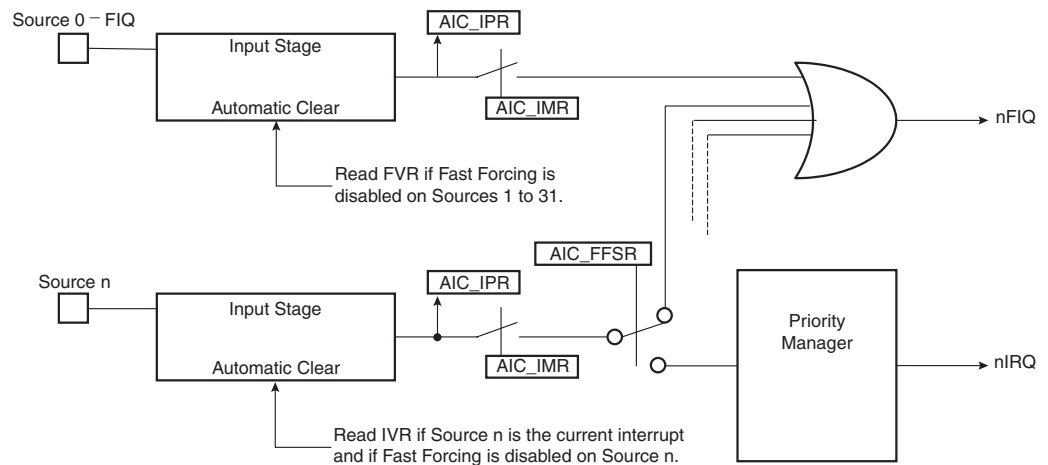
The Fast Interrupt Vector Register (AIC\_FVR) reads the contents of the Source Vector Register 0 (AIC\_SVR0), whatever the source of the fast interrupt may be. The read of the FVR does not clear the Source 0 when the fast forcing feature is used and the interrupt source should be cleared by writing to 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 52. Fast Forcing**



## 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.

## 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.

## 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.



## Advanced Interrupt Controller (AIC) User Interface

### 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 an  $\pm 4$ -Kbyte offset.

**Table 46.** Advanced Interrupt Controller (AIC) 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	Fast Interrupt Vector Register	AIC_FVR	Read-only	0x0
0x108	Interrupt Status Register	AIC_ISR	Read-only	0x0
0x10C	Interrupt Pending Register	AIC_IPR	Read-only	0x0 <sup>(1)</sup>
0x110	Interrupt Mask Register	AIC_IMR	Read-only	0x0
0x114	Core Interrupt Status Register	AIC_CISR	Read-only	0x0
0x118	Reserved	---	---	---
0x11C	Reserved	---	---	---
0x120	Interrupt Enable Command Register	AIC_IECR	Write-only	---
0x124	Interrupt Disable Command Register	AIC_IDCR	Write-only	---
0x128	Interrupt Clear Command Register	AIC_ICCR	Write-only	---
0x12C	Interrupt Set Command Register	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	AIC_FFER	Write-only	---
0x144	Fast Forcing Disable Register	AIC_FFDR	Write-only	---
0x148	Fast Forcing Status Register	AIC_FFSR	Read-only	0x0

Note: 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.

## 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
0	0	Level Sensitive
0	1	Edge Triggered
1	0	Level Sensitive
1	1	Edge Triggered

## 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.

## AIC Interrupt Vector Register

**Register Name:** AIC\_IVR

**Access Type:** Read-only

**Reset Value:** 0

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.

## AIC FIQ Vector Register

**Register Name:** AIC\_FVR

**Access Type:** Read-only

**Reset Value:** 0

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 Fast Interrupt Vector Register reads the value stored in AIC\_SPU.

## AIC Interrupt Status Register

**Register Name:** AIC\_ISR

**Access Type:** Read-only

**Reset Value:** 0

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

## AIC Interrupt Pending Register

**Register Name:** AIC\_IPR

**Access Type:** Read-only

**Reset Value:** 0

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.



## AIC Interrupt Mask Register

**Register Name:** AIC\_IMR

**Access Type:** Read-only

**Reset Value:** 0

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.

## AIC Core Interrupt Status Register

**Register Name:** AIC\_CISR

**Access Type:** Read-only

**Reset Value:** 0

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-
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.



## 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.

## 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.

## 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.

## 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.

## 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.

## AIC Spurious Interrupt Vector Register

**Register Name:** AIC\_SPU

**Access Type:** Read/Write

**Reset Value:** 0

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.

## AIC Debug Control Register

**Register Name:** AIC\_DEBUG

**Access Type:** Read/Write

**Reset Value:** 0

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

## 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.

## 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.

## 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.

## Clock Generator

### Description

The Clock Generator is made up of 1 PLL, a Main Oscillator, an RC Oscillator and 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 “Power Management Controller (PMC) User Interface” on page 161. However, the Clock Generator registers are named CKGR\_.

### Slow Clock RC Oscillator

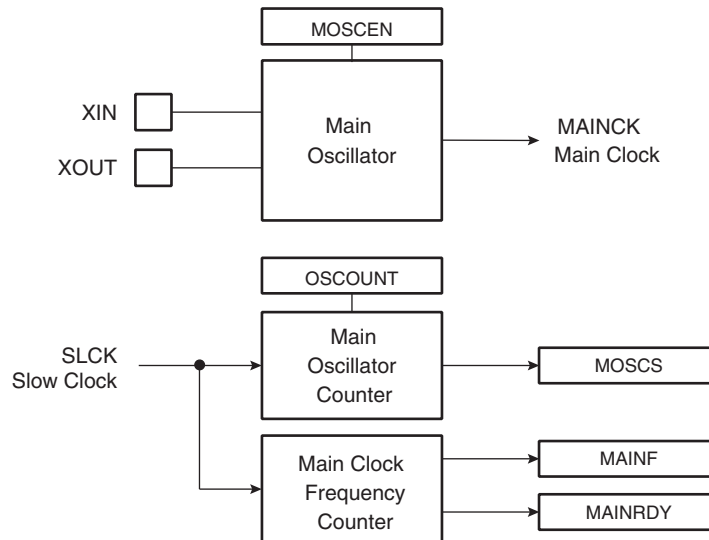
The slow clock is the output of the RC Oscillator and is the only clock considered permanent in a system that includes the Power Management Controller. It is mandatory in the operations of the PMC.

The user has to take the possible drifts of the RC Oscillator into account. More details are given in the DC Characteristics section of the product datasheet.

### Main Oscillator

Figure 53 shows the Main Oscillator block diagram.

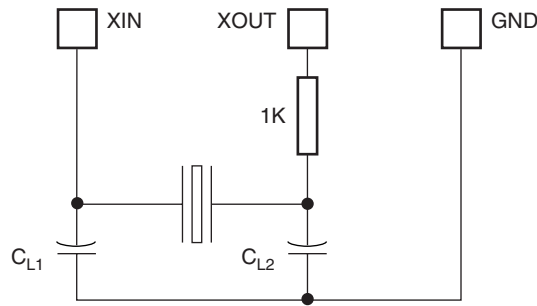
**Figure 53.** Main Oscillator Block Diagram



### 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 54. The 1 kΩ resistor is only required for crystals with frequencies lower than 8 MHz. The oscillator contains 25 pF capacitors on each XIN and XOUT pin. Consequently, CL1 and CL2 can be removed when a crystal with a load capacitance of 12.5 pF is used. For further details on the electrical characteristics of the Main Oscillator, see the DC Characteristics section of the product datasheet.

**Figure 54.** Typical Crystal Connection



**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.

**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.

**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.

**Main Oscillator Bypass**

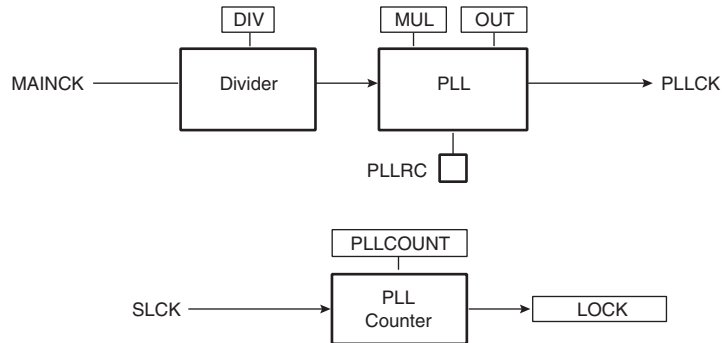
The user can input a clock on the device instead of connecting a crystal. In this case, the user has to provide the external clock signal on the XIN pin. The input characteristics of the XIN pin under these conditions are given in the product electrical characteristics section. The programmer has to 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.



## 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 55 shows the block diagram of the divider and PLL block.

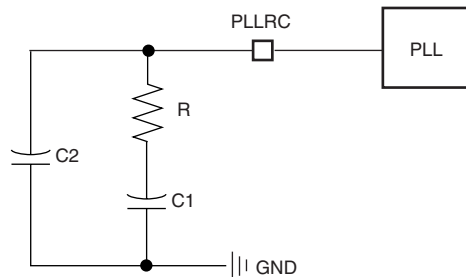
**Figure 55.** Divider and PLL Block Diagram



## PLL Filter

The PLL requires connection to an external second-order filter through the PLLRC pin. Figure 56 shows a schematic of these filters.

**Figure 56.** 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.

## 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.

# Power Management Controller (PMC)

## 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.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.

## 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.

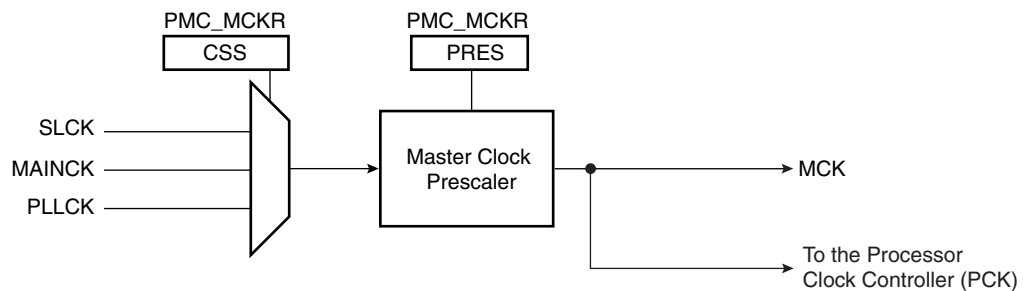
The Master Clock is selected from one of the clocks provided by the Clock Generator. Selecting the Slow Clock provides a Slow Clock signal to the whole device. Selecting the Main Clock saves power consumption of the PLL.

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 57.** Master Clock Controller



## 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.

## 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.

## Programmable Clock Output Controller

The PMC controls 3 signals to be output on external pins PCKx. Each signal can be independently programmed via the PMC\_PCKx registers.

PCKx can be independently selected between the Slow clock, 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.

## 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 \* OSCOUNT / 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.

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 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.

All parameters in PMC\_MCKR can be programmed in a single write operation. 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 "Clock Switching Waveforms" on page 159.

Code Example:

```
write_register(PMC_MCKR, 0x00000011)
```

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, 3 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, 8 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.

### Clock Switching Details

#### Master Clock Switching Timings

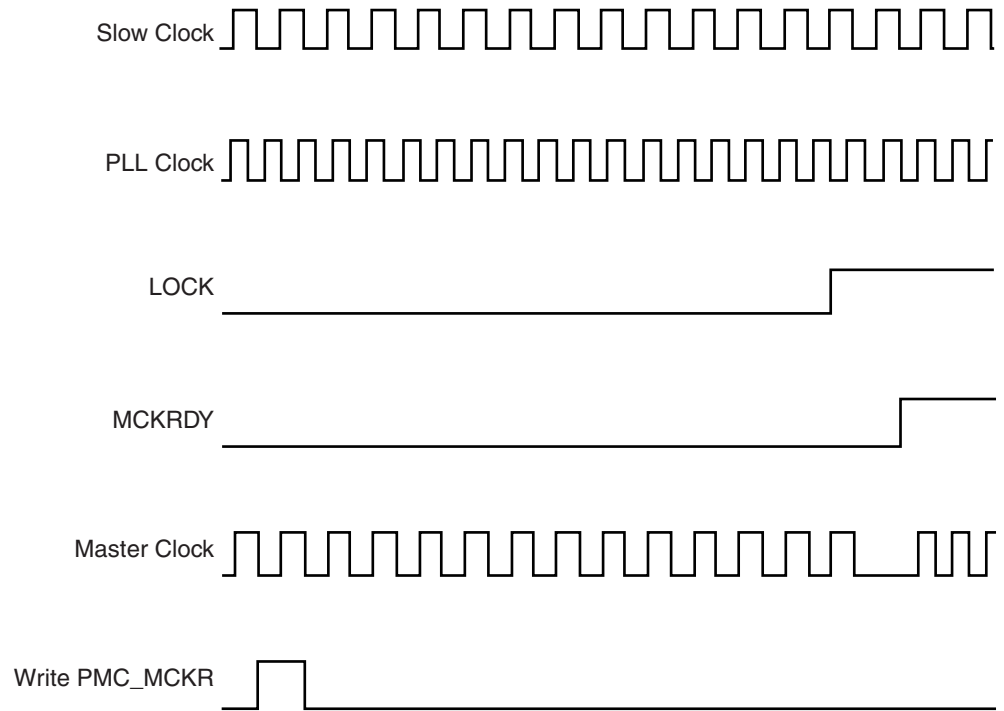
Table 47 gives the worst case timing required for the Master Clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the new selected clock has to be added.

**Table 47.** Clock Switching Timings (Worst Case)

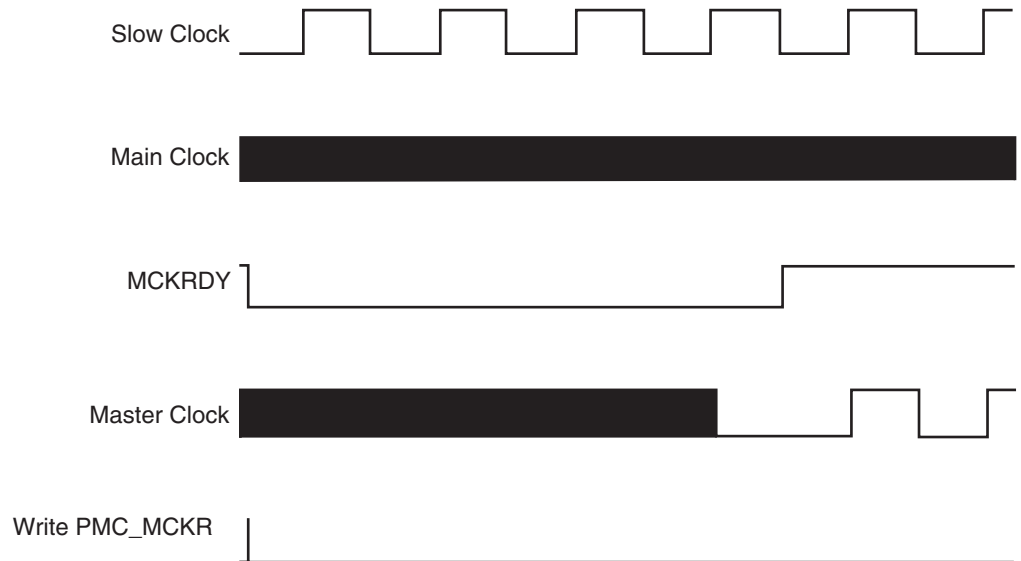
<b>To</b>	<b>From</b>	<b>Main Clock</b>	<b>SLCK</b>	<b>PLL Clock</b>
<b>Main Clock</b>		–	4 x SLCK + 2.5 x Main Clock	3 x PLL Clock + 4 x SLCK + 1 x Main Clock
<b>SLCK</b>		0.5 x Main Clock + 4.5 x SLCK	–	3 x PLL Clock + 5 x SLCK
<b>PLL Clock</b>		0.5 x Main Clock + 4 x SLCK + PLLCOUNT x SLCK + 2.5 x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

## Clock Switching Waveforms

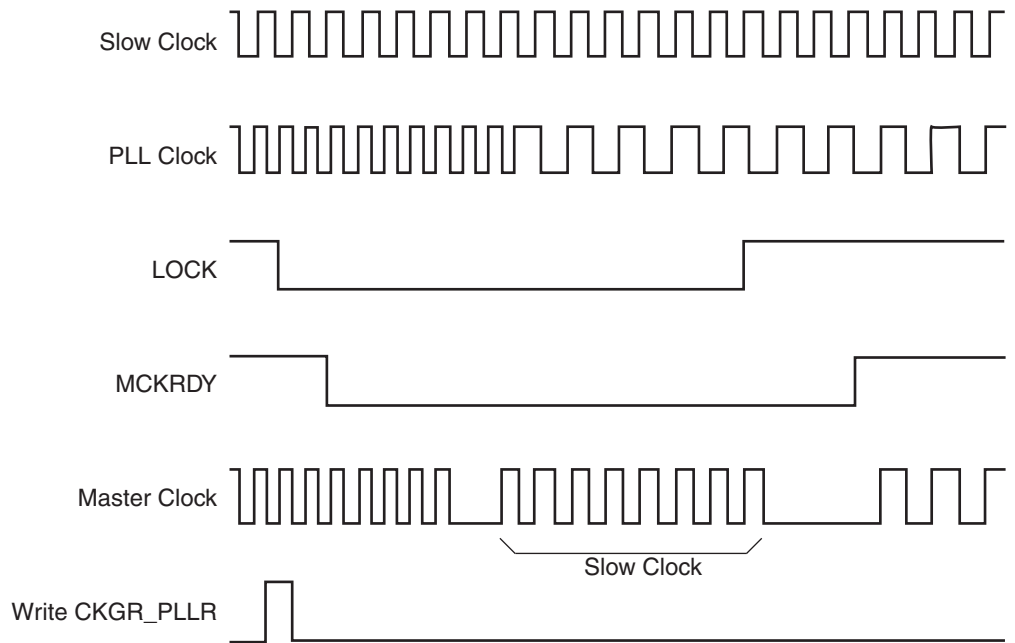
**Figure 58.** Switch Master Clock from Slow Clock to PLL Clock



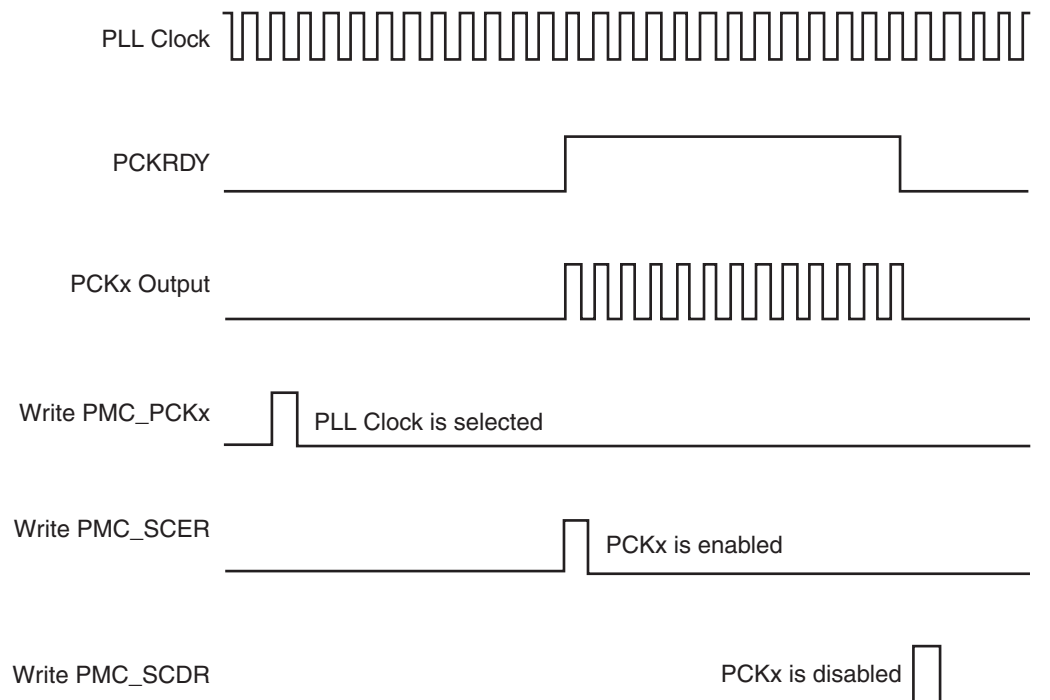
**Figure 59.** Switch Master Clock from Main Clock to Slow Clock



**Figure 60.** Change PLL Programming



**Figure 61.** Programmable Clock Output Programming





## Power Management Controller (PMC) User Interface

**Table 48.** Power Management Controller (PMC) 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
0x001C	Reserved	–	–	–
0x0020	Main Oscillator Register	CKGR_MOR	Read/Write	0x0
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read-only	–
0x0028	Reserved	–	–	–
0x002C	PLL Register	CKGR_PLLR	Read/Write	0x3F00
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0
0x0034	Reserved	–	–	–
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	0x18
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0
0x0070 - 0x00FC	Reserved	–	–	–

### 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
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	PCK

- **PCK: Processor Clock Enable**

0 = No effect.

1 = Enables the Processor clock.

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

0 = No effect.

1 = Enables the corresponding Programmable Clock output.

## 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
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	PCK

- **PCK: Processor Clock Disable**

0 = No effect.

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

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

0 = No effect.

1 = Disables the corresponding Programmable Clock output.

### 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
–	–	–	–	–	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	PCK

- **PCK: Processor Clock Status**

0 = The Processor clock is disabled.

1 = The Processor clock is enabled.

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

0 = The corresponding Programmable Clock output is disabled.

1 = The corresponding Programmable Clock output is enabled.

## 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: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

## 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.

### 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.

### 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.

- **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.

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

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

## 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.



**PMC Clock Generator PLL Register**

**Register Name:** CKGR\_PLLR

**Access Type:** Read/Write

31	30	29	28	27	26	25	24
-					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**

OUT		PLL Clock Frequency Range
0	0	Refer to the DC Characteristics section of the product datasheet
0	1	Reserved
1	0	Refer to the DC Characteristics section of the product datasheet
1	1	Reserved

- **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.



**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: Master Clock Prescaler**

PRES			Master 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



**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			Master 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

## 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.

## 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.

### 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.

### 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.

## Debug Unit (DBGU)

### 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.

Finally, the Debug Unit features a Force NTRST capability that enables the software to decide whether to prevent access to the system via the In-circuit Emulator. This permits protection of the code, stored in ROM.

## Block Diagram

Figure 62. Debug Unit Functional Block Diagram

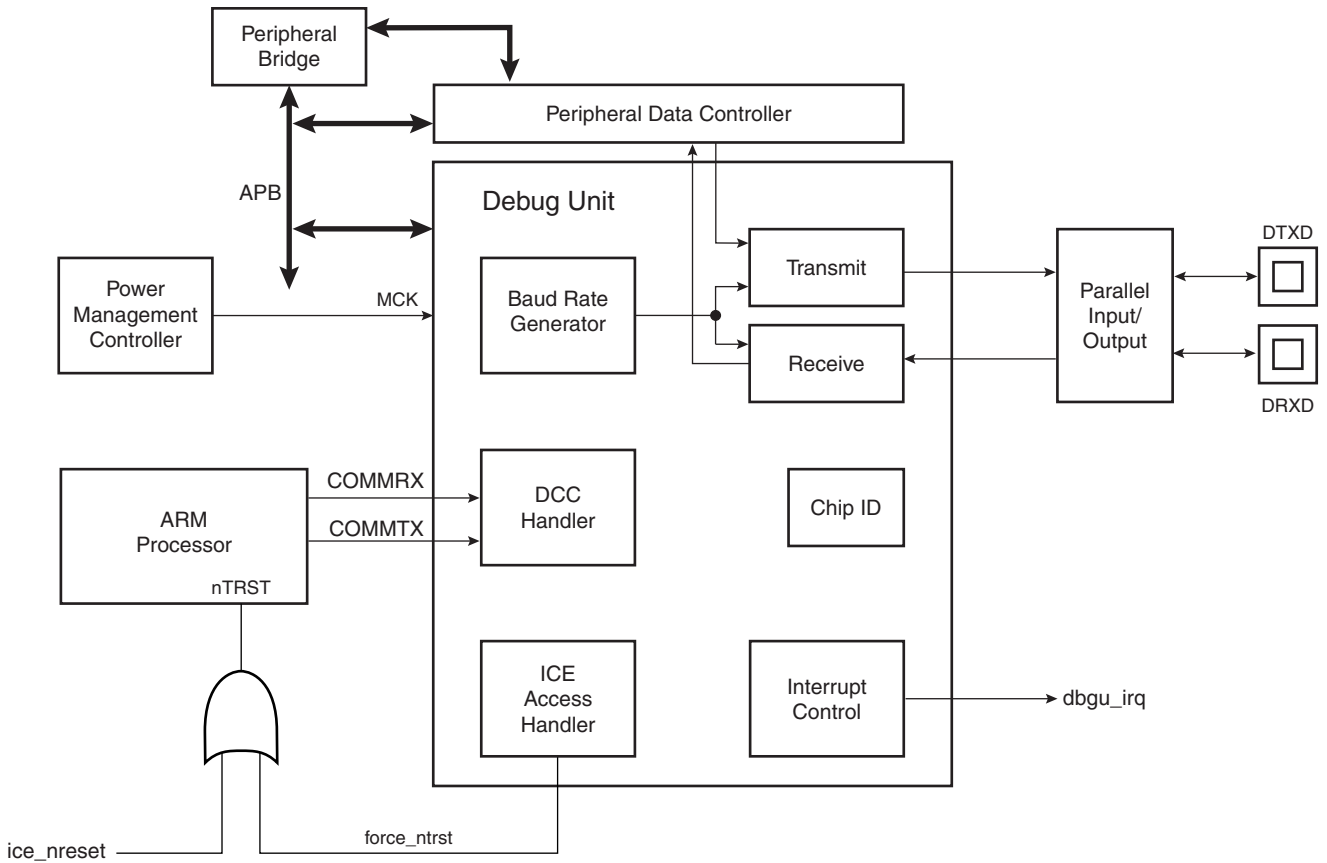
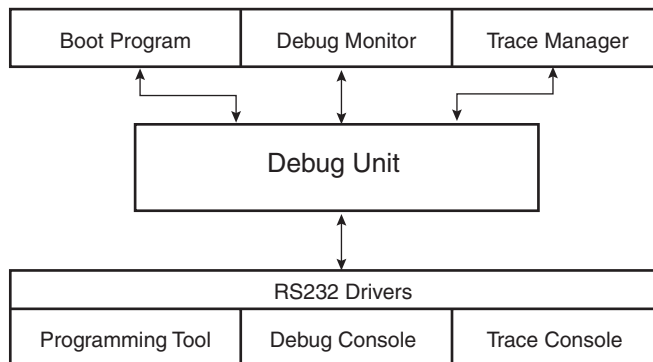


Table 49. Debug Unit Pin Description

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

Figure 63. Debug Unit Application Example



## Product Dependencies

### 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.

### 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.

### 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 62. This sharing requires the programmer to determine the source of the interrupt when the source 1 is triggered.

## 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.

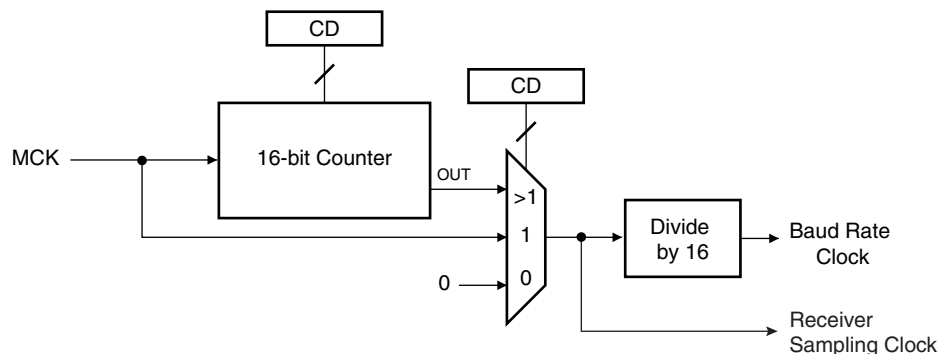
### Baud Rate Generator

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 64.** Baud Rate Generator



## Receiver

### 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.

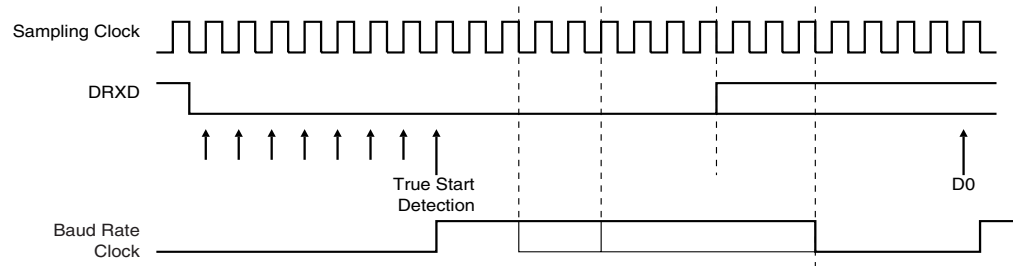
### Start Detection and Data Sampling

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.

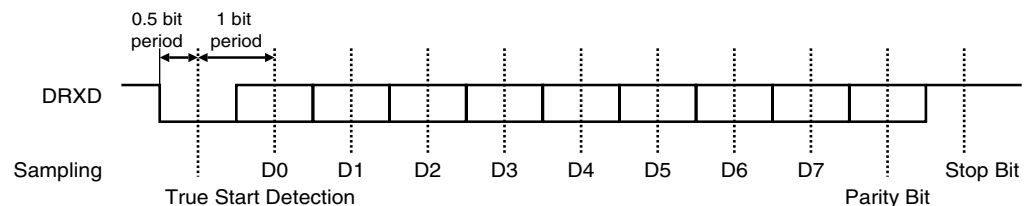
Each subsequent bit is sampled 16 cycles (1-bit period) after the previous one.

**Figure 65.** Start Bit Detection



**Figure 66.** Character Reception

Example: 8-bit, parity enabled 1 stop

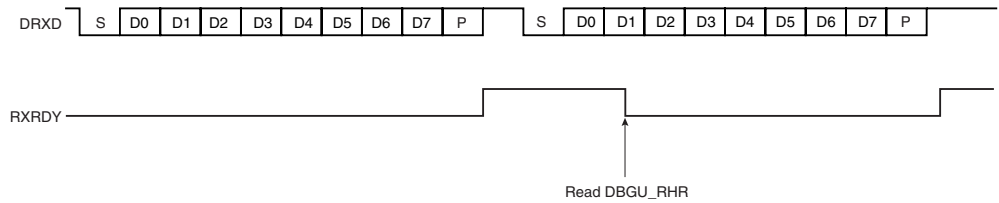


### Receiver Ready

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.



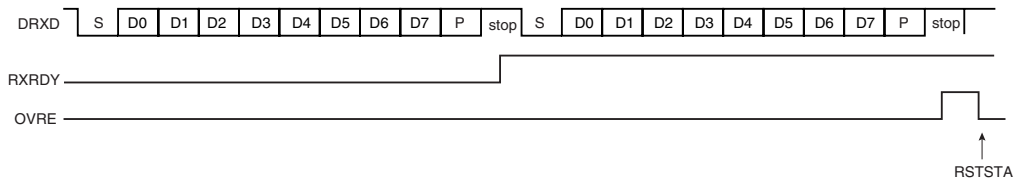
**Figure 67. Receiver Ready**



## 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.

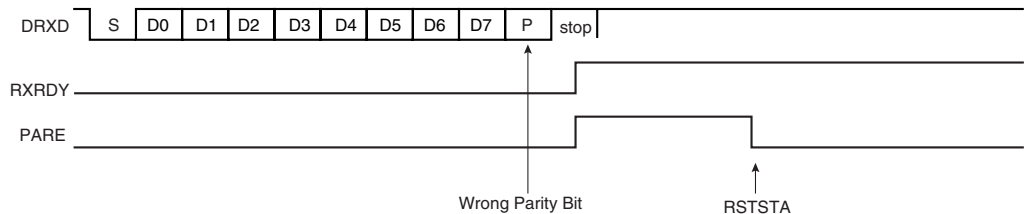
**Figure 68. Receiver Overrun**



## 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.

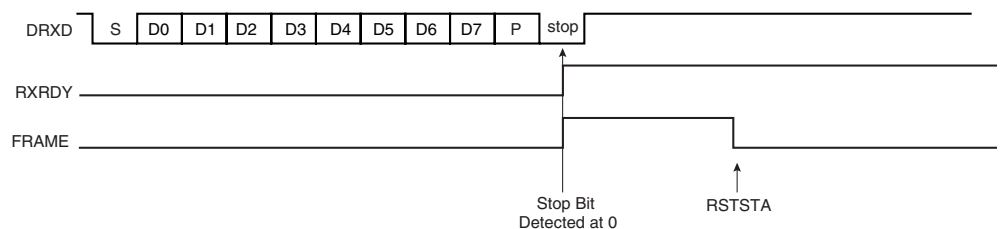
**Figure 69. Parity Error**



## 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 70. Receiver Framing Error**



## Transmitter

### 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.

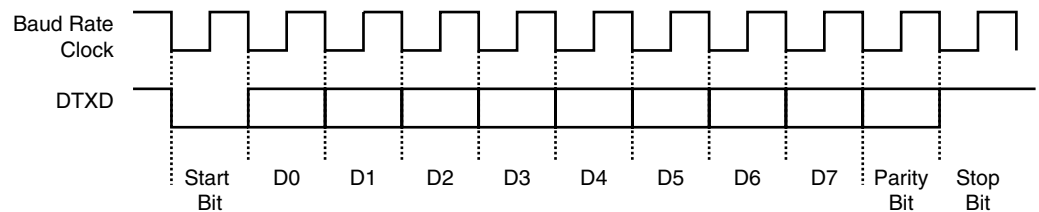
The programmer can also put the transmitter in its reset state by writing the `DBGU_CR` with the bit `RSTTX` at 1. This immediately stops the transmitter, whether or not it is processing characters.

### 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 71.** Character Transmission

Example: Parity enabled

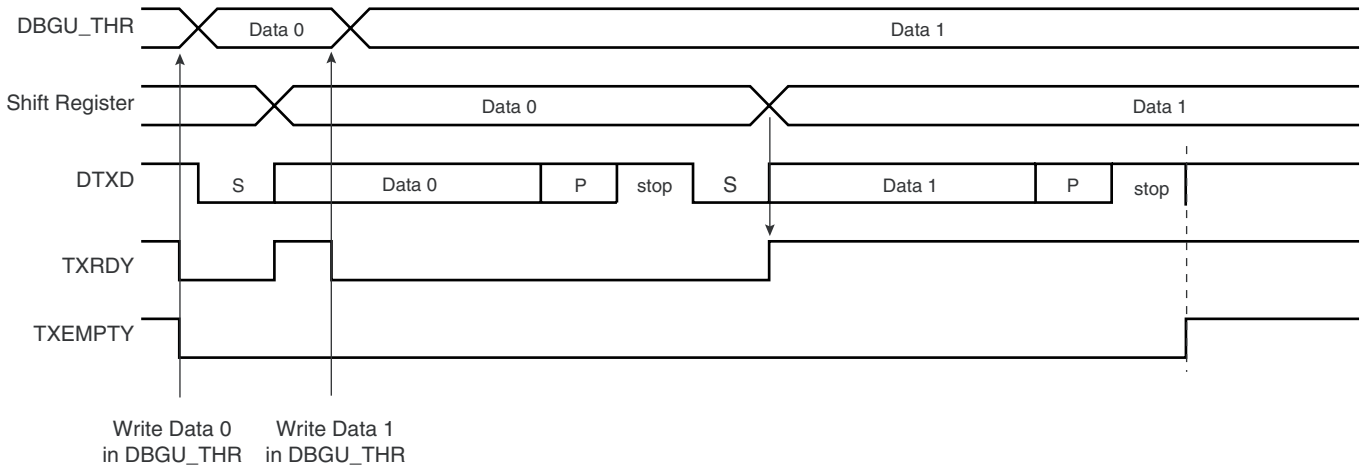


### Transmitter Control

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 72. Transmitter Control**



## 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`.

## Test Modes

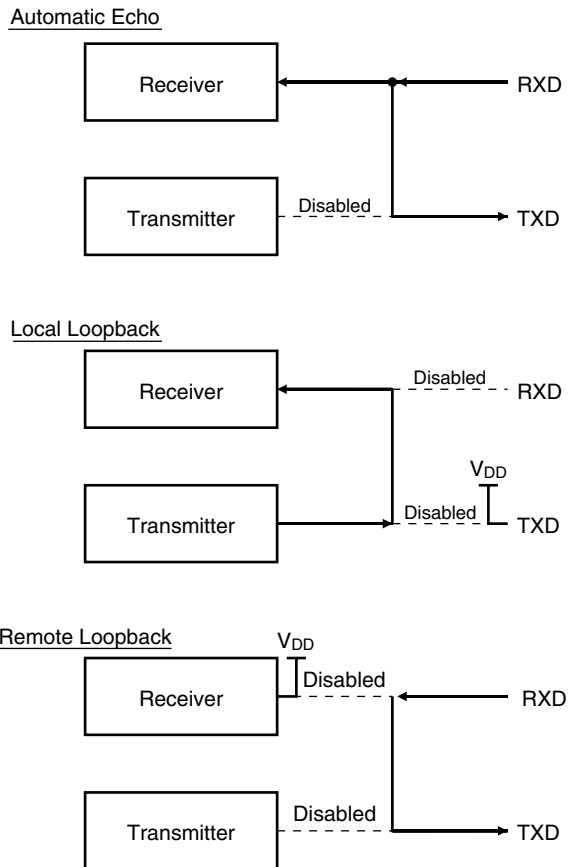
The Debug Unit supports three tests modes. These modes of operation are programmed by using the field `CHMODE` (Channel Mode) in the mode register `DBGU_MR`.

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 73. Test Modes**



## 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.

## 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.

## ICE Access Prevention

The Debug Unit allows blockage of access to the system through the ARM processor's ICE interface. This feature is implemented via the register Force NTRST (DBGU\_FNR), that allows assertion of the NTRST signal of the ICE Interface. Writing the bit FNTRST (Force NTRST) to 1 in this register prevents any activity on the TAP controller.

On standard devices, the bit FNTRST resets to 0 and thus does not prevent ICE access.

This feature is especially useful on custom ROM devices for customers who do not want their on-chip code to be visible.

## Debug Unit User Interface

**Table 50.** 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	Force NTRST Register	DBGU_FNR	Read/Write	0x0
0x004C - 0x00FC	Reserved	–	–	–
0x0100 - 0x0124	PDC Area	–	–	–

## 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.

## 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



## 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.

## 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.

## 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.

## 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.



## 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.

## 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.

## 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	$MCK / (CD \times 16)$



## 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	Reserved
0	1	0	1	4K bytes
0	1	1	0	Reserved
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	
0x40	0100 0000	AT91x40 Series
0x63	0110 0011	AT91x63 Series
0x55	0101 0101	AT91x55 Series
0x42	0100 0010	AT91x42 Series
0x92	1001 0010	AT91x92 Series
0x34	0011 0100	AT91x34 Series
0x70	0111 0000	AT91SAM7Sxx and AT91SAM7Axx Series
0x71	0111 0001	AT91SAM7Xxx Series
0x72	0111 0010	AT91SAM7Exx Series
0x73	0111 0011	AT91SAM7Lxx Series
0x19	0001 1001	AT91SAM9xx 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.

## 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.

## Debug Unit Force NTRST Register

**Name:** DBGU\_FNR

**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
–	–	–	–	–	–	–	FNTRST

- FNTRST: Force NTRST**

0 = NTRST of the ARM processor's TAP controller is driven by the ice\_nreset signal.

1 = NTRST of the ARM processor's TAP controller is held low.



## Parallel Input/Output Controller (PIO)

### 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.

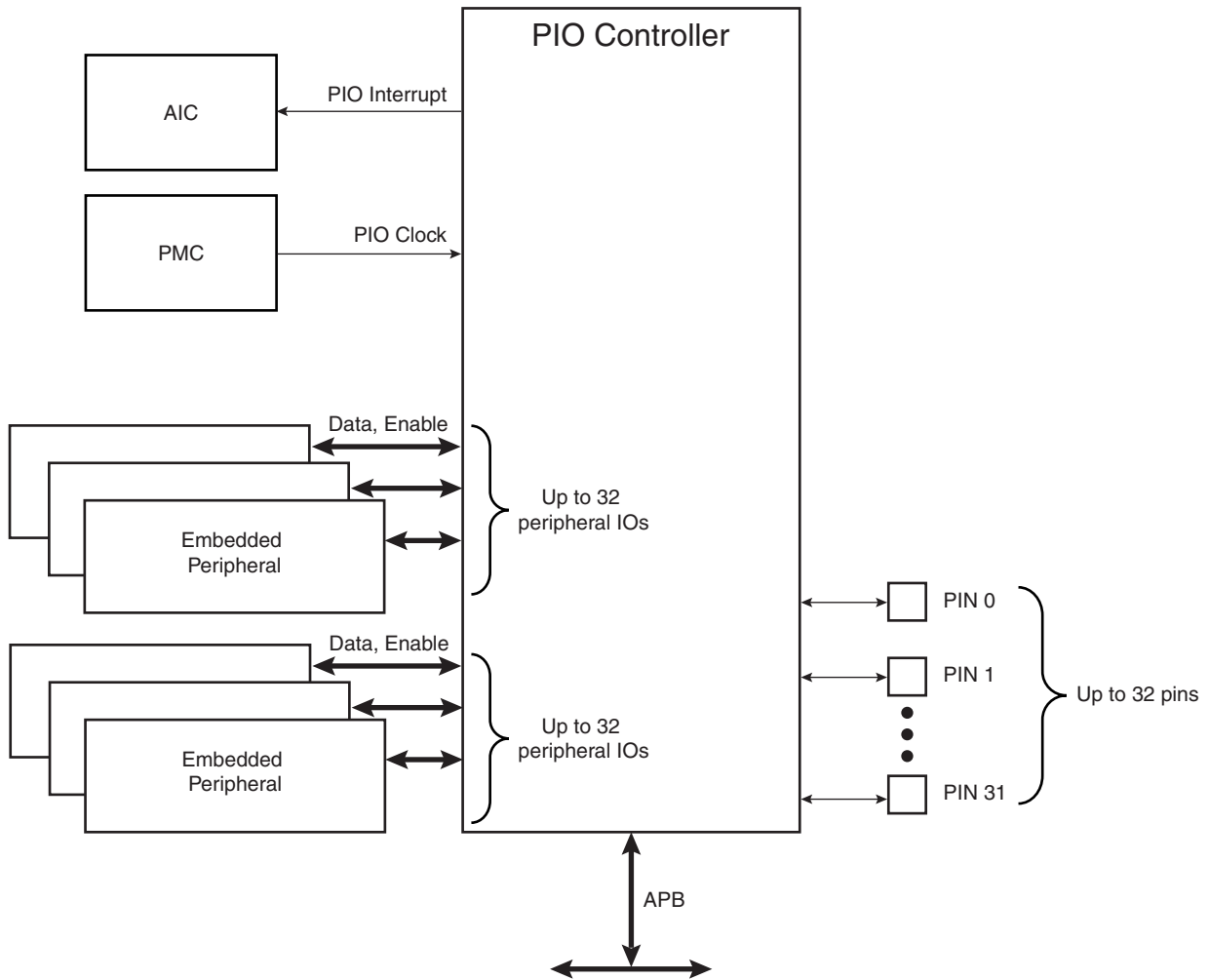
Each I/O line of the PIO Controller features:

- 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 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.

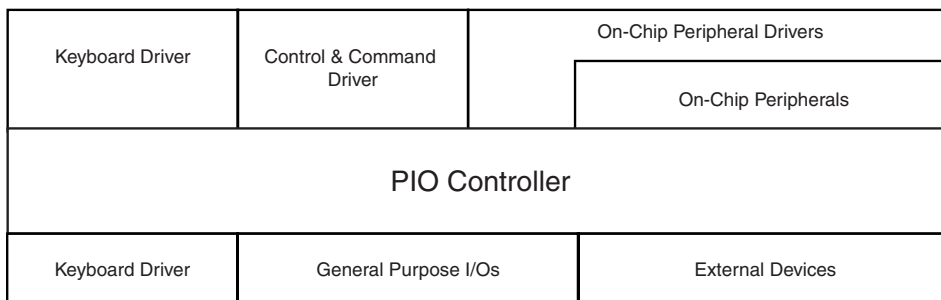
## Block Diagram

Figure 74. Block Diagram



## Application Block Diagram

Figure 75. Application Block Diagram



## Product Dependencies

### 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.

### 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.

### 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.

### Interrupt Generation

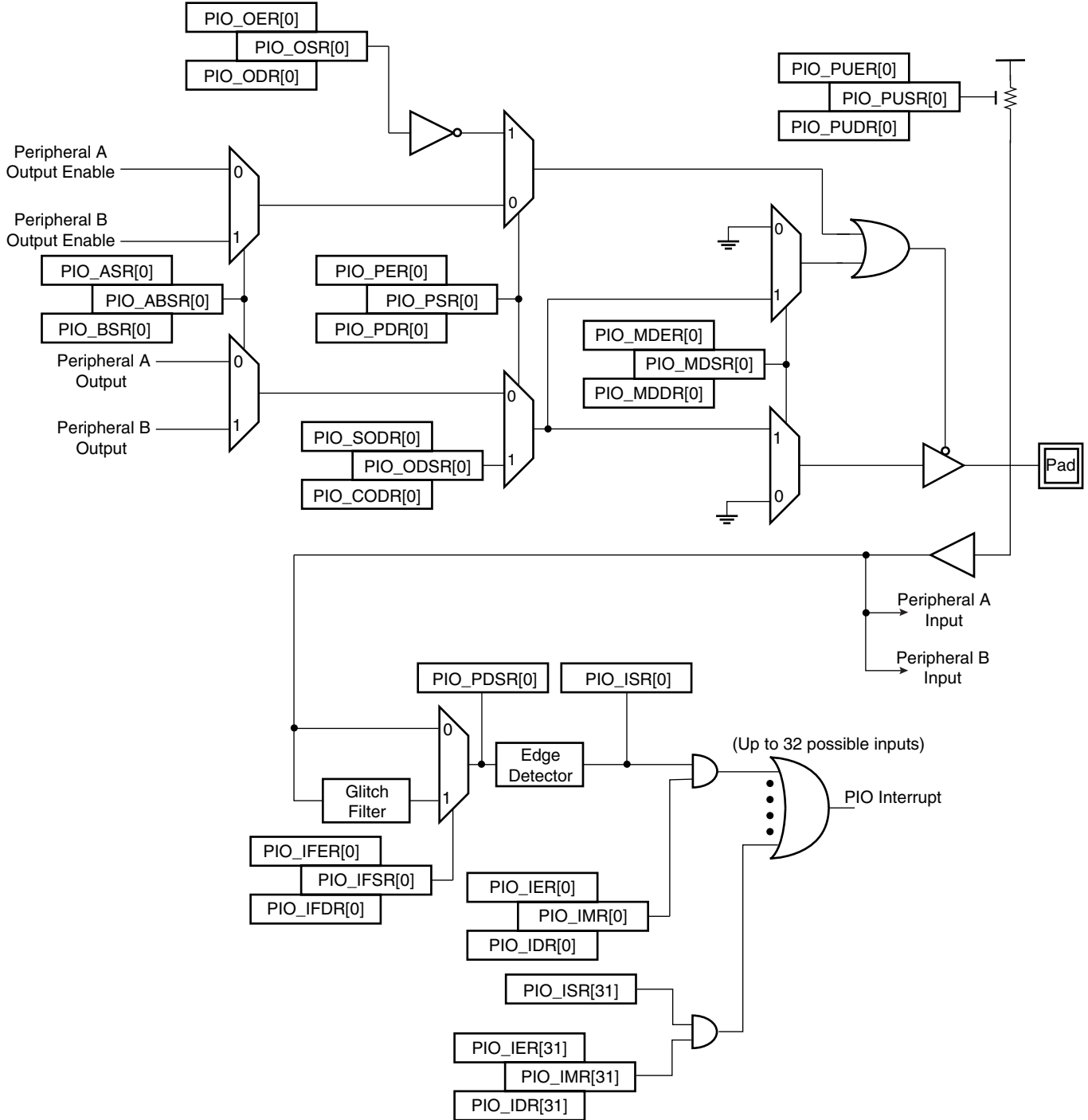
For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources 2 to 31. Refer to the PIO Controller peripheral identifier in the product description to identify the interrupt sources dedicated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

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

## 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 76. In this description each signal shown represents but one of up to 32 possible indexes.

Figure 76. I/O Line Control Logic



### Pull-up Resistor Control

Each I/O line is designed with an embedded pull-up resistor. The value of this resistor is about 100 k $\Omega$  (see the product electrical characteristics for more details about this value). 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.

## 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.

## 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.

## 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\_PDR (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.

### 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\_OSWSR 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.

### 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.

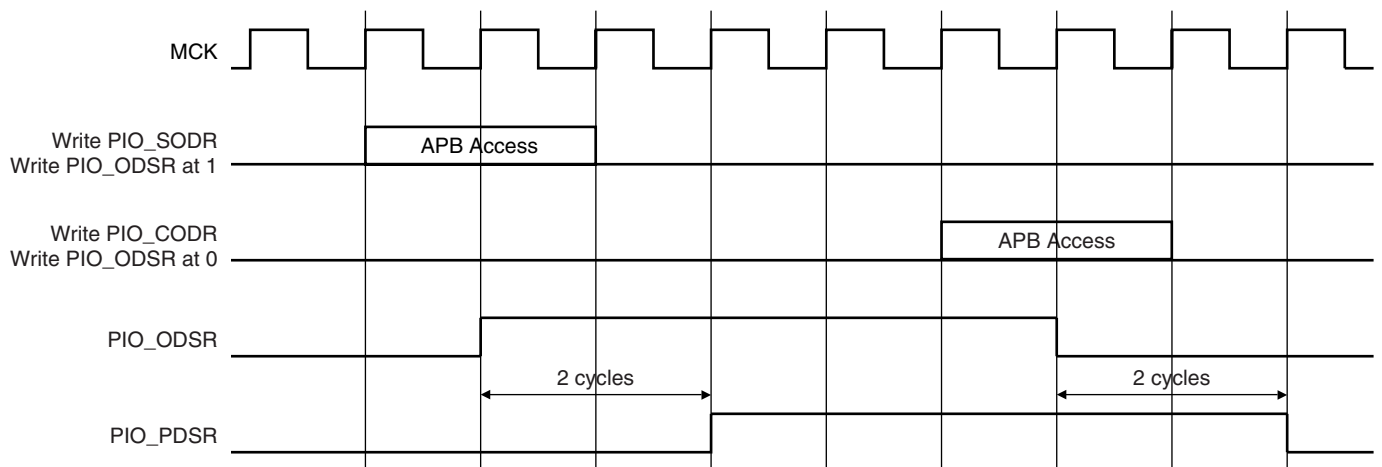
The Multi Drive feature is controlled by PIO\_MDER (Multi-driver Enable Register) and PIO\_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO\_MDSR (Multi-driver Status Register) indicates 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.

### Output Line Timings

Figure 77 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\_OSWSR is set. Figure 77 also shows when the feedback in PIO\_PDSR is available.

Figure 77. Output Line Timings



### Inputs

The level on each I/O line can be read through PIO\_PDSR (Peripheral 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.

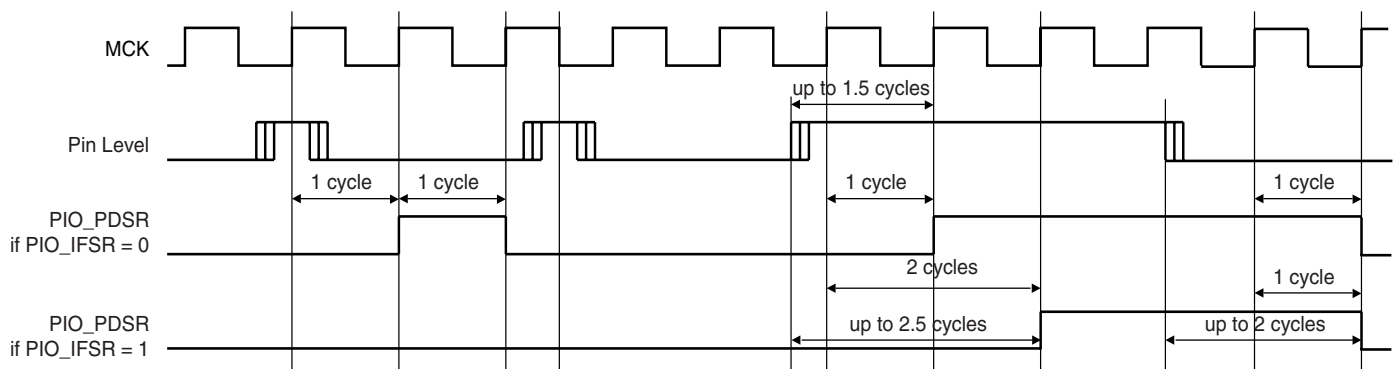
## Input Glitch Filtering

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 78.

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 78.** Input Glitch Filter Timing



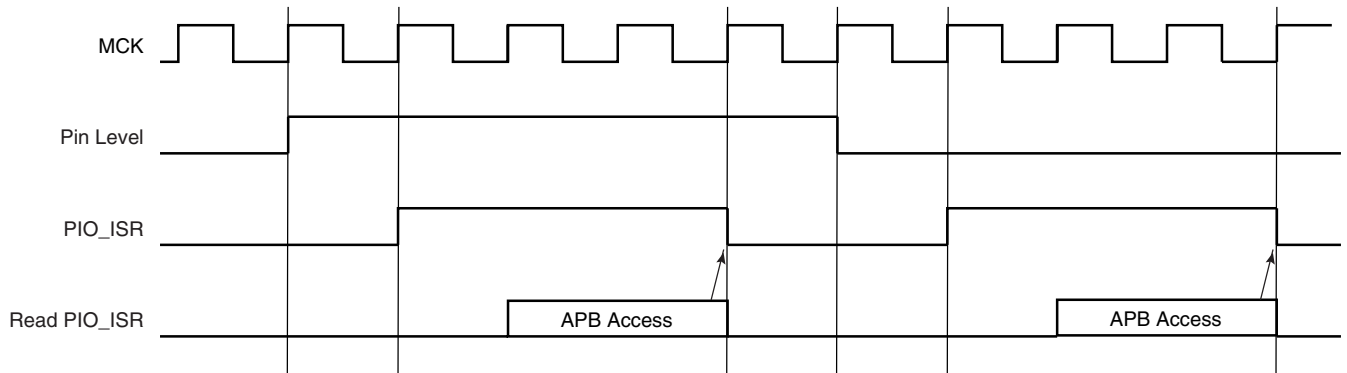
## Input Change Interrupt

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 79. Input Change Interrupt Timings**



## I/O Lines Programming Example

The programming example as shown in Table 51 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 51.** 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



## Parallel Input/Output Controller (PIO) User Interface

Each I/O line controlled by the PIO Controller is associated with a bit in each of the PIO Controller User Interface registers. Each register is 32 bits wide. If a parallel I/O line is not defined, writing to the corresponding bits has no effect. Undefined bits read zero. If the I/O line is not multiplexed with any peripheral, the I/O line is controlled by the PIO Controller and PIO\_PSR returns 1 systematically.

**Table 52.** Parallel Input/Output Controller (PIO) 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 52.** Parallel Input/Output Controller (PIO) 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 - 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 - 0x00FC	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.





## 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).

## 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).



## 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).

## 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.



## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.



## 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.

## 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.

## 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.

## 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.



## 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.

## 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.



## 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.

## 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.



## 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.

## 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.

## 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.

## PIO Peripheral A Select Register

**Name:** PIO\_ASX

**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.



## 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.

## 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.

## 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.

## 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.



## 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.

## Serial Peripheral Interface (SPI)

### 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).

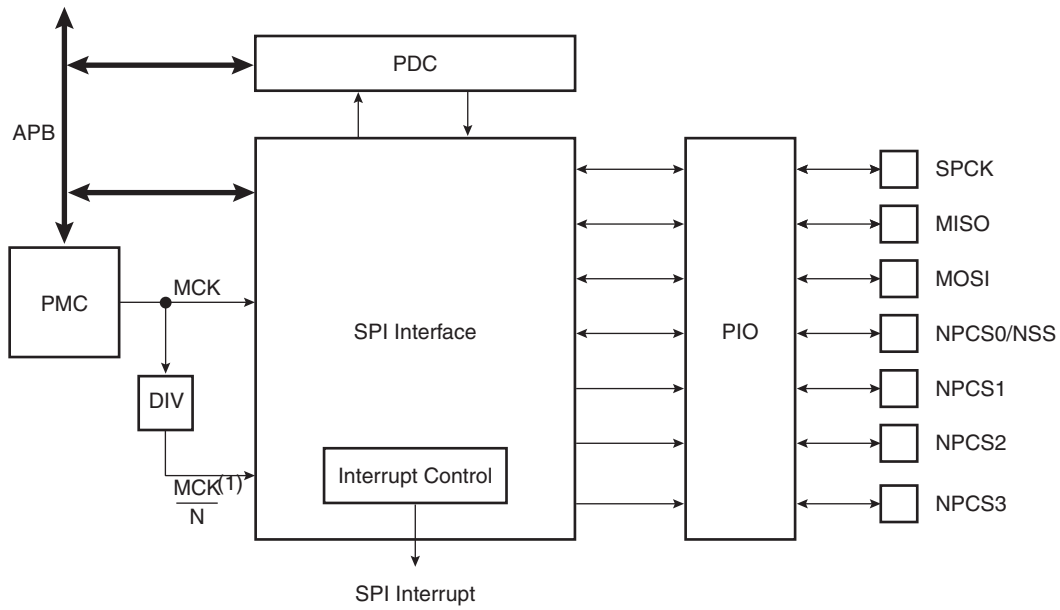
The SPI system consists of two data lines and two control lines:

- 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.



## Block Diagram

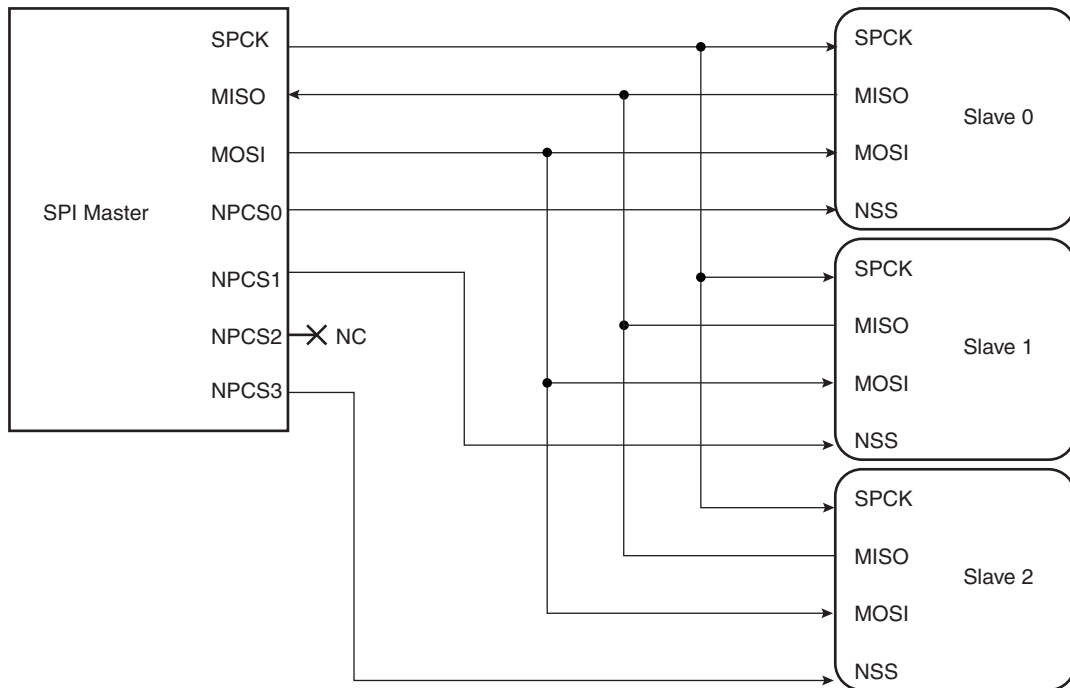
Figure 80. Block Diagram



Note: 1.  $N = 32$

## Application Block Diagram

Figure 81. Application Block Diagram: Single Master/Multiple Slave Implementation





## Signal Description

**Table 53.** 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

## Product Dependencies

### 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.

### 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.

### 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.

## Functional Description

### 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.

### 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.

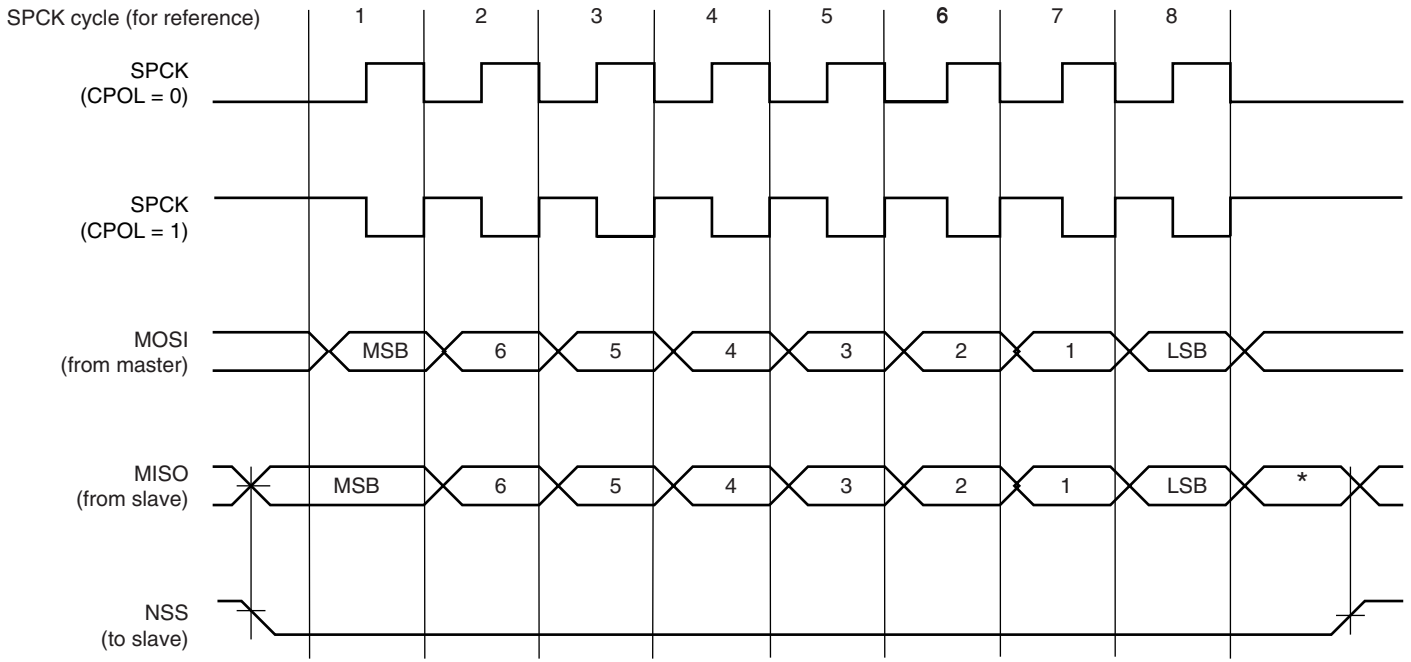
Table 54 shows the four modes and corresponding parameter settings.

**Table 54.** SPI Bus Protocol Mode

SPI Mode	CPOL	CPHA
0	0	1
1	0	0
2	1	1
3	1	0

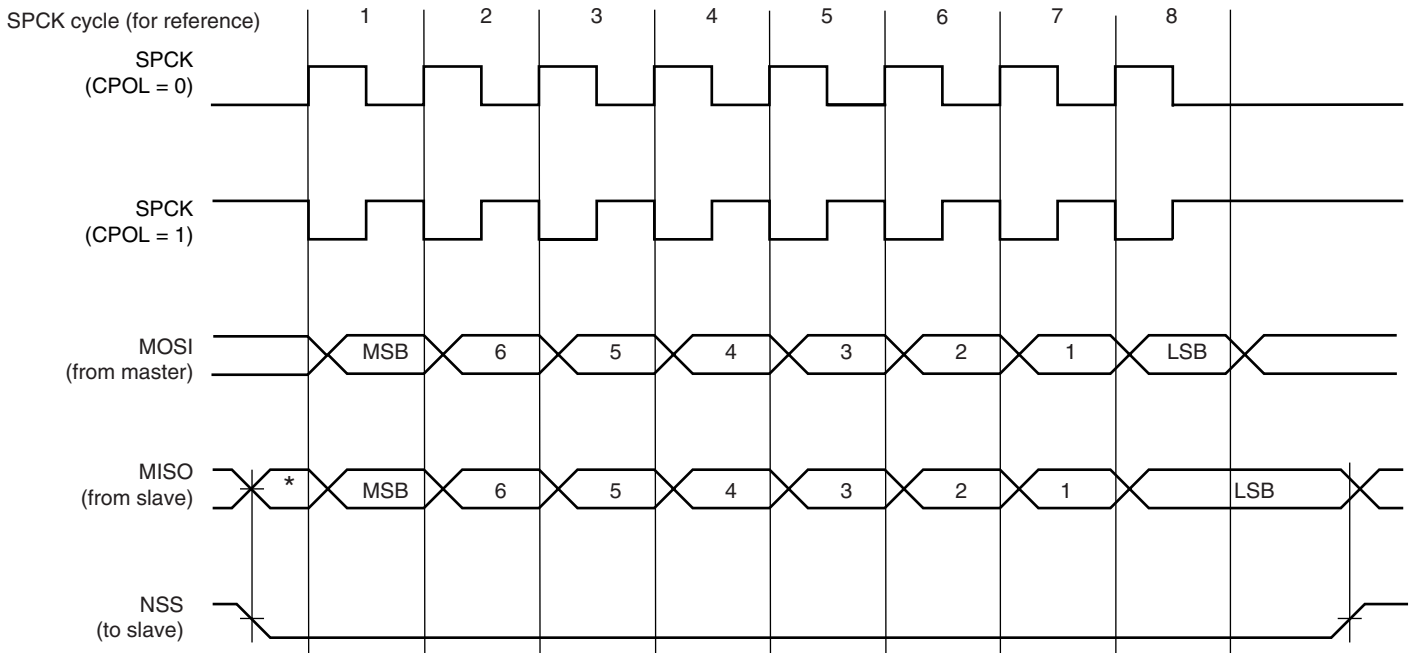
Figure 82 and Figure 83 show examples of data transfers.

**Figure 82. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)**



\* Not defined, but normally MSB of previous character received.

**Figure 83. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)**



\* Not defined but normally LSB of previous character transmitted.

## 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.

No transfer is started when writing into the SPI\_TDR if the PCS field does not select a slave. The PCS field is set by writing the SPI\_TDR in variable mode, or the SPI\_MR in fixed mode, depending on the value of PCS field.

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.

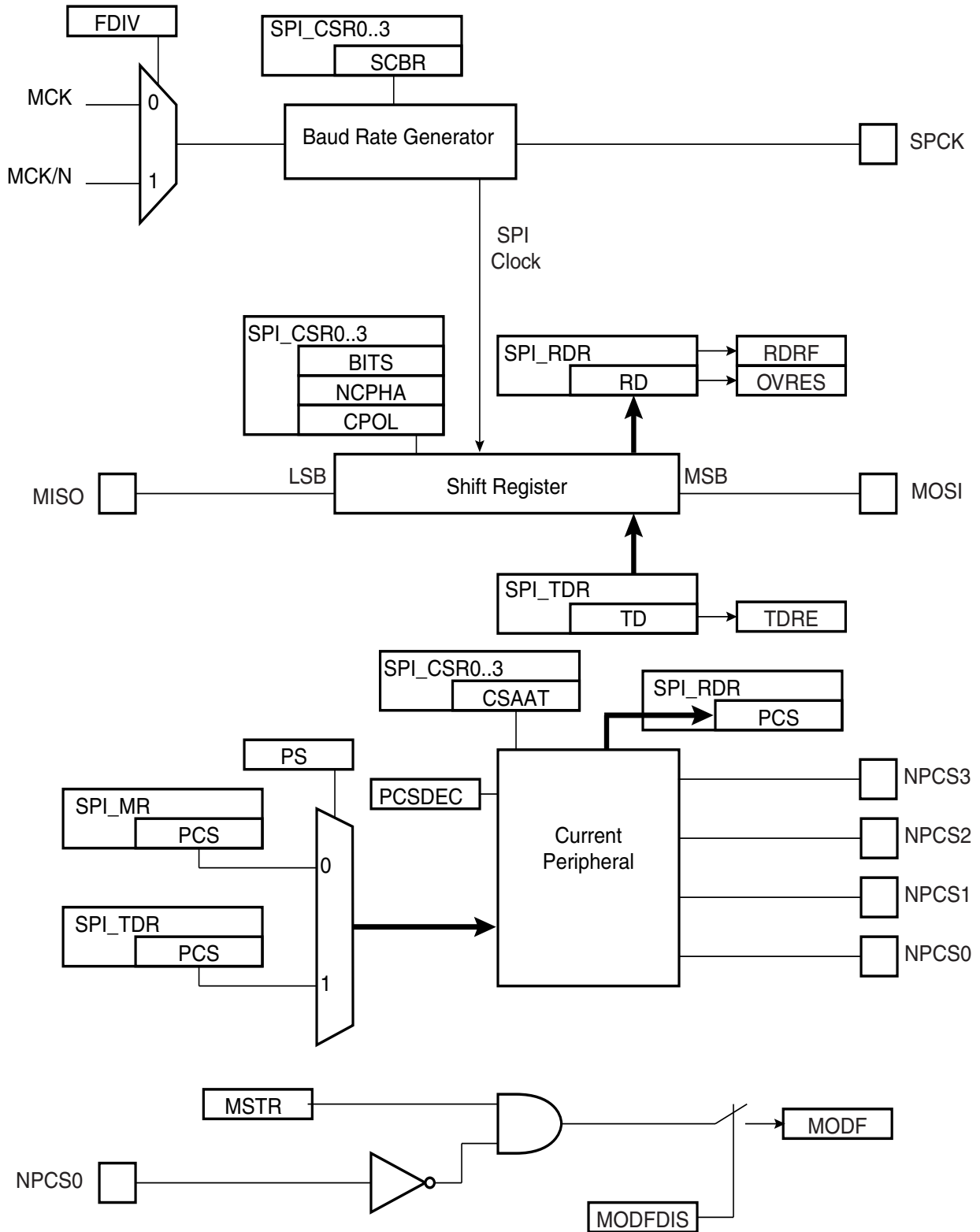
The transfer of received data from the Shift Register in SPI\_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI\_SR). When the received data is read, the RDRF bit is cleared.

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, no data is loaded in SPI\_RDR. The user has to read the status register to clear the OVRES bit.

Figure 84 on page 229 shows a block diagram of the SPI when operating in Master Mode. Figure 85 on page 230 shows a flow chart describing how transfers are handled.

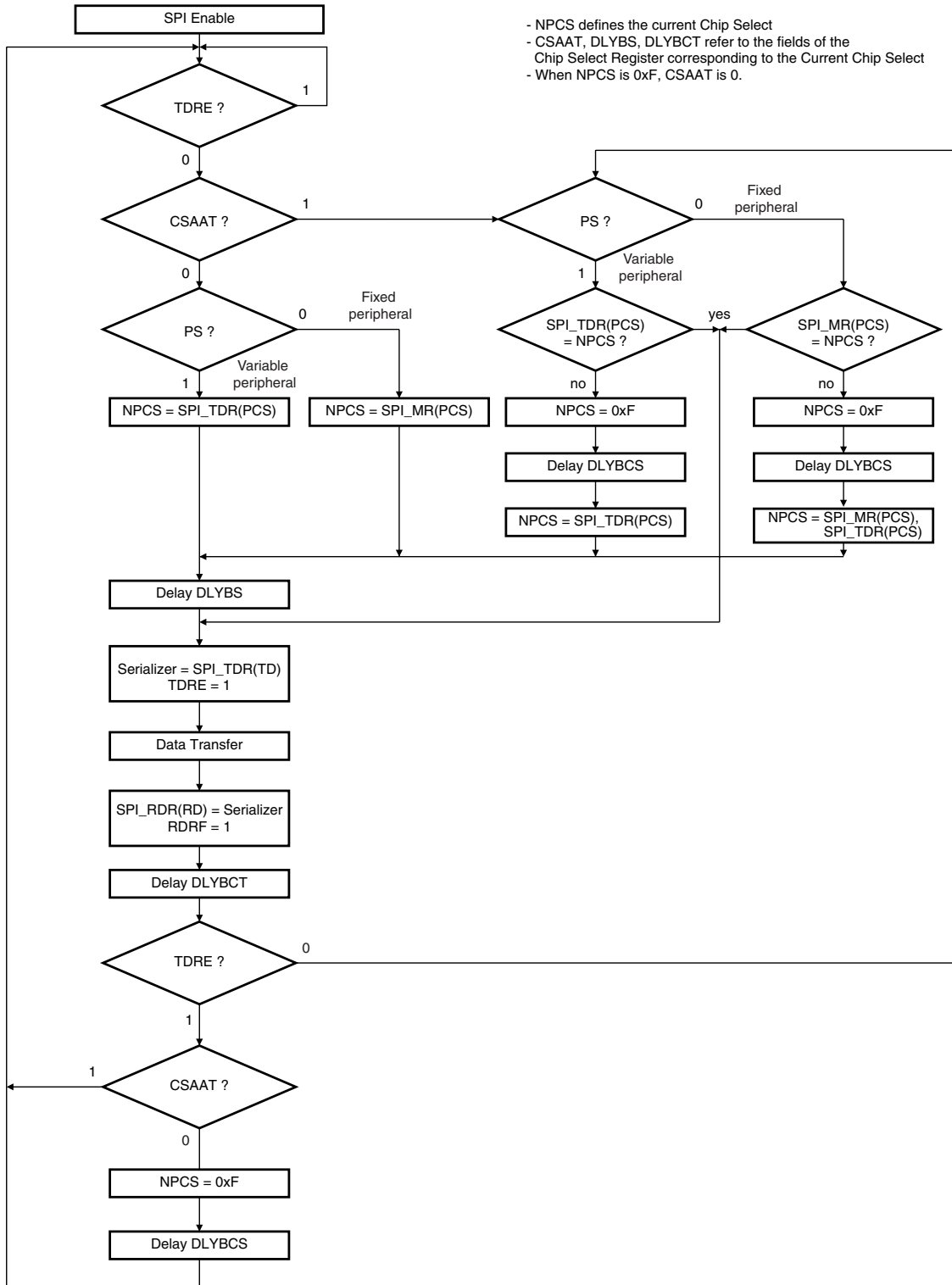
## Master Mode Block Diagram

Figure 84. Master Mode Block Diagram



## Master Mode Flow Diagram

Figure 85. Master Mode Flow Diagram S



## Clock Generation

The SPI Baud rate clock is generated by dividing the Master Clock (MCK) or the Master Clock divided by 32, by a value between 2 and 255. The selection between Master Clock or Master Clock divided by N is done by the FDIV value set in the Mode Register

This allows a maximum operating baud rate at up to Master Clock/2 and a minimum operating baud rate of MCK divided by 255\*32.

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.

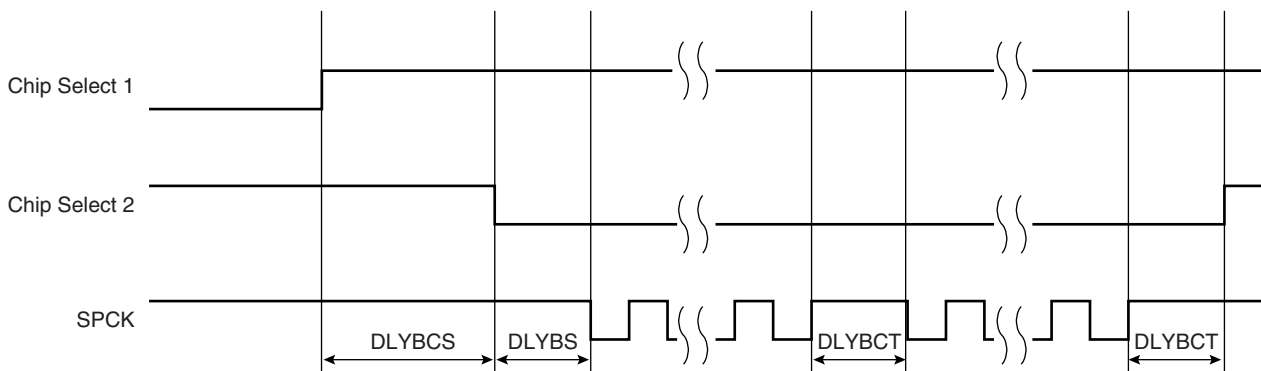
## Transfer Delays

Figure 86 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 86.** Programmable Delays



## 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 fields of the Chip Select Registers have 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.

### 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\_CR0 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.

### Peripheral Deselection

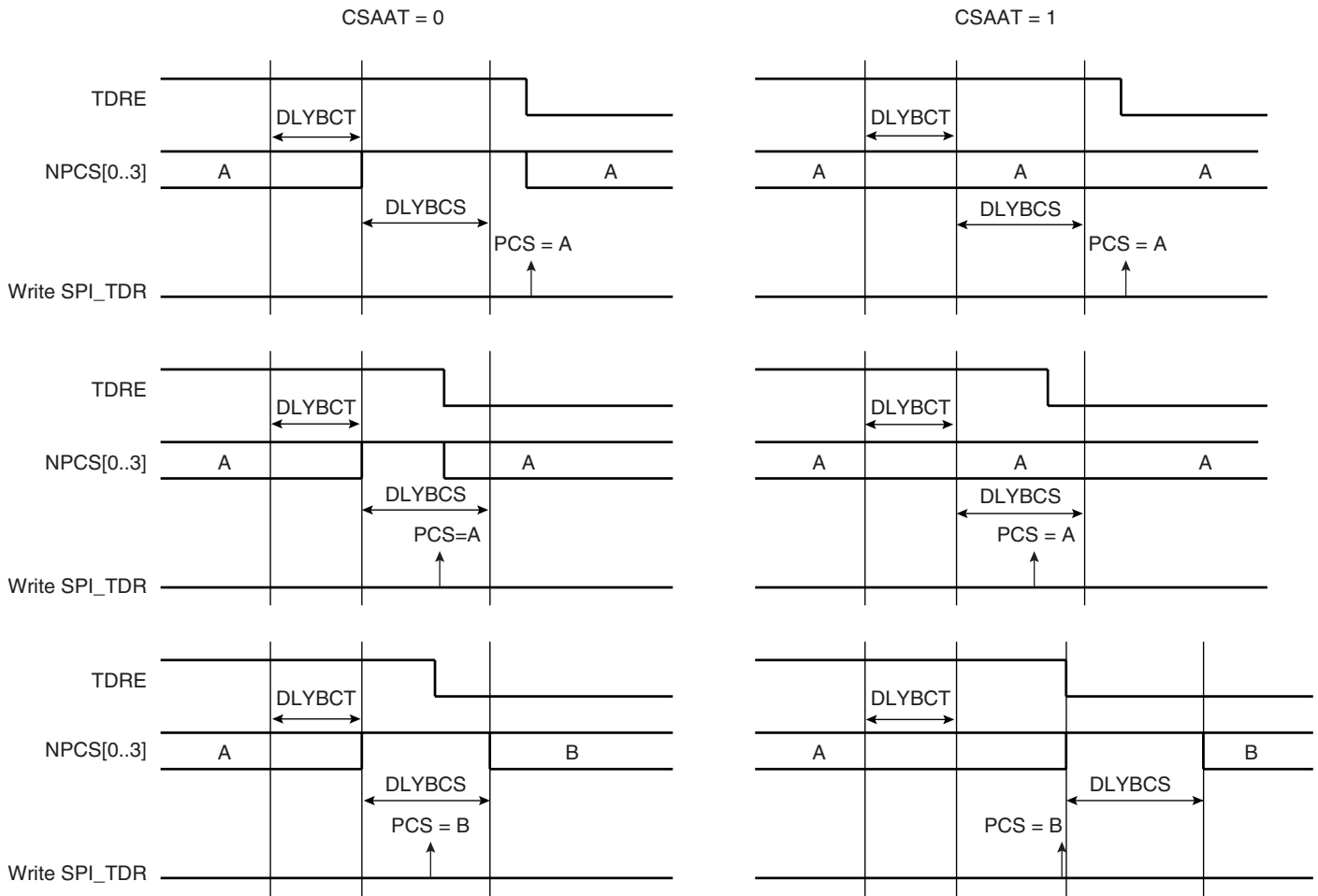
When operating normally, as soon as the transfer of the last data written in SPI\_TDR is completed, the NPCS lines all rise. This might lead to runtime error if the processor is too long in responding to an interrupt, and thus might lead to difficulties for interfacing with some serial peripherals requiring 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 87 shows different peripheral deselection cases and the effect of the CSAAT bit.



**Figure 87. Peripheral Deselection**



## 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 NPCSO/NSS signal. As this pin is generally configured in open-drain, it is important that a pull up resistor is connected on the NPCSO line, so that a high level is guaranteed and no spurious mode fault is detected.

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).

## 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.

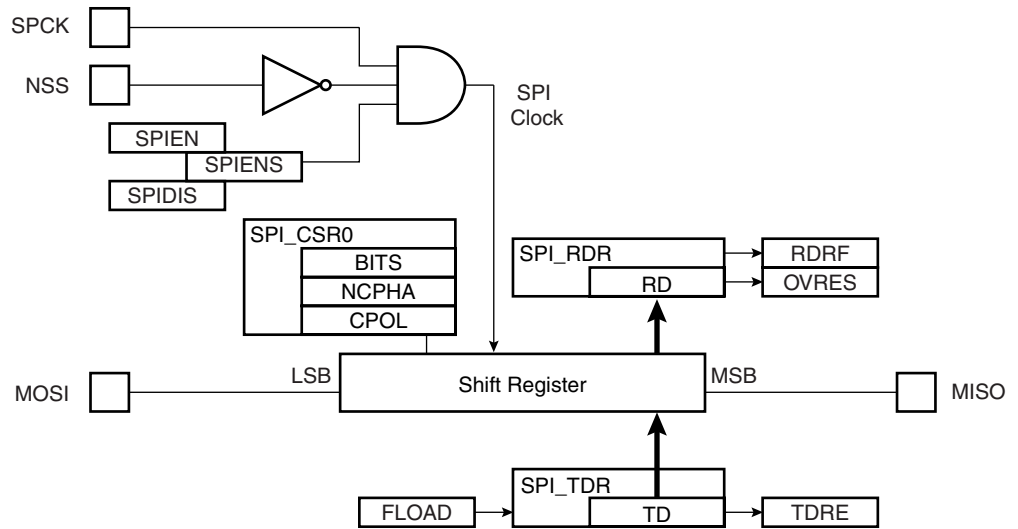
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI\_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets 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 88 shows a block diagram of the SPI when operating in Slave Mode.

**Figure 88.** Slave Mode Functional Block Diagram



## Serial Peripheral Interface (SPI) User Interface

**Table 55.** Serial Peripheral Interface (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 - 0x00FC	Reserved	–	–	–
0x100 - 0x124	Reserved for the PDC			

## 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.

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.

- **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.

## 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	FDIV	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 16 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 15.

- **FDIV: Clock Selection**

0 = The SPI operates at MCK.

1 = The SPI operates at MCK/N.

- **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.

- **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 (or 6\*N MCK periods if FDIV is set) will be inserted by default.

Otherwise, the following equation determines the delay:

If FDIV is 0:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

If FDIV is 1:

$$\text{Delay Between Chip Selects} = \frac{DLYBCS \times N}{MCK}$$

## 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.

## 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.



## 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 or SPI\_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in SPI\_RCR or SPI\_RNCR.

- **ENDTX: End of TX buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in SPI\_TCR or SPI\_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in SPI\_TCR or SPI\_TNCR.

- **RXBUFF: RX Buffer Full**

0 = SPI\_RCR or SPI\_RNCR has a value other than 0.

1 = Both SPI\_RCR and SPI\_RNCR has a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = SPI\_TCR or SPI\_TNCR has a value other than 0.

1 = Both SPI\_TCR and SPI\_TNCR has 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.

## 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.

## 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.

## 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.



## 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:

If FDIV is 0:

$$\text{SPCK Baudrate} = \frac{MCK}{SCBR}$$

If FDIV is 1:

$$\text{SPCK Baudrate} = \frac{MCK}{(N \times SCBR)}$$

Note: N = 32

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:

If FDIV is 0:

$$\text{Delay Before SPCK} = \frac{DLYBS}{MCK}$$

If FDIV is 1:

$$\text{Delay Before SPCK} = \frac{N \times DLYBS}{MCK}$$

Note: N = 32

- **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:

If FDIV is 0:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK} + \frac{SCBR}{2MCK}$$

If FDIV is 1:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times N \times DLYBCT}{MCK} + \frac{N \times SCBR}{2MCK}$$

Note: N = 32





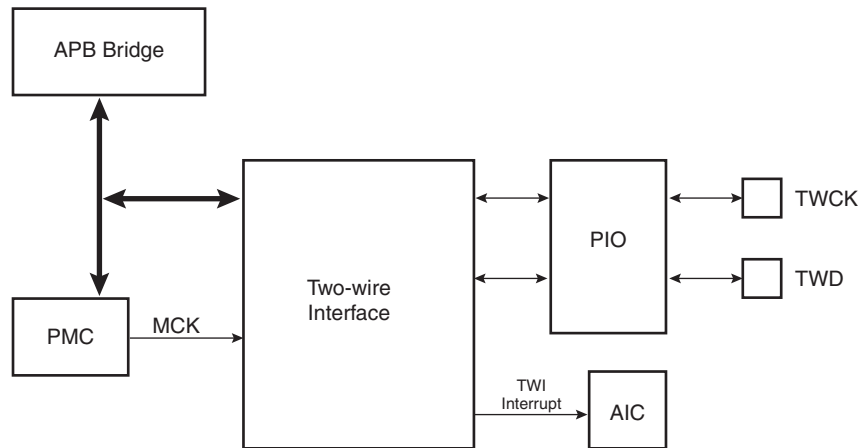
## Two-wire Interface (TWI)

### 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.

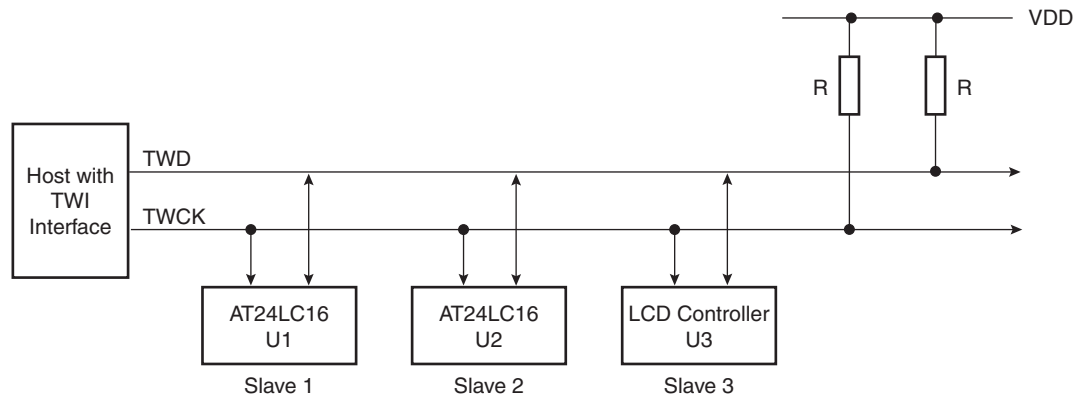
### Block Diagram

Figure 89. Block Diagram



### Application Block Diagram

Figure 90. Application Block Diagram



## Product Dependencies

### I/O Lines Description

**Table 56.** 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 90 on page 249). 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.
  
- Enable the peripheral clock.

### Power Management

The TWI interface may be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the TWI clock.

### 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.

## Functional Description

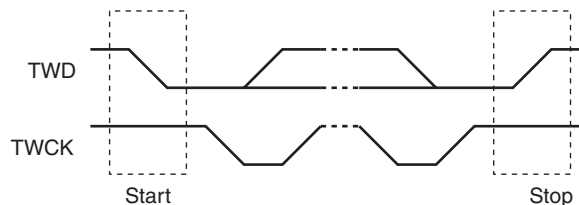
### Transfer Format

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 92 on page 251).

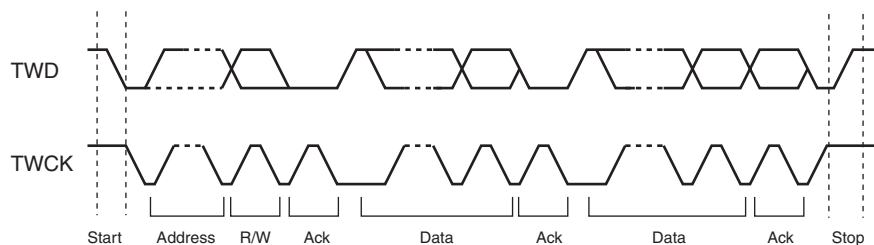
Each transfer begins with a START condition and terminates with a STOP condition (see Figure 91 on page 251).

- 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 91.** START and STOP Conditions



**Figure 92.** Transfer Format



### 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.

### 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).

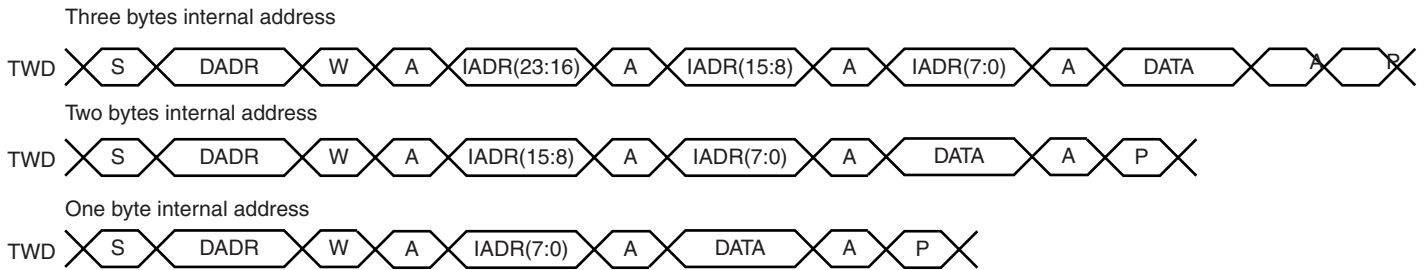
The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse, the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the **NAK** bit 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 94 below). The master generates a stop condition to end the transfer.

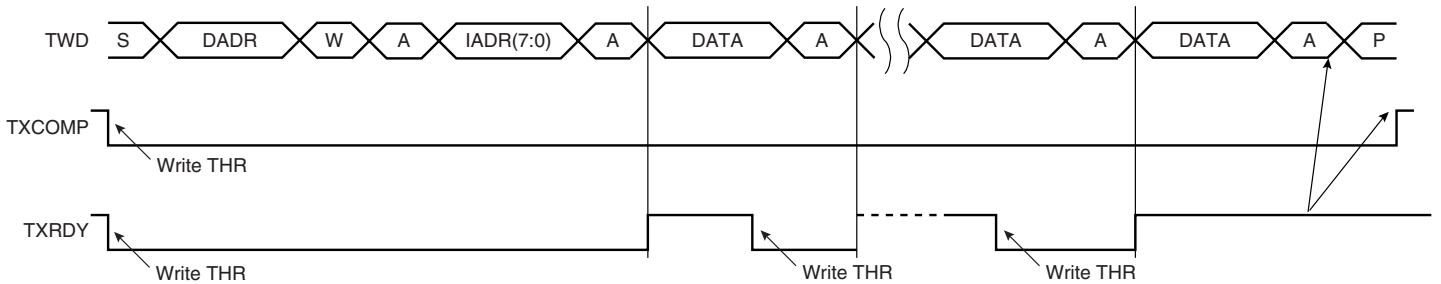
The read sequence begins by setting the START bit. When the RXRDY bit is set in the status register, a character has been received in the receive-holding register (TWI\_RHR). The RXRDY bit is reset when reading the TWI\_RHR.

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).

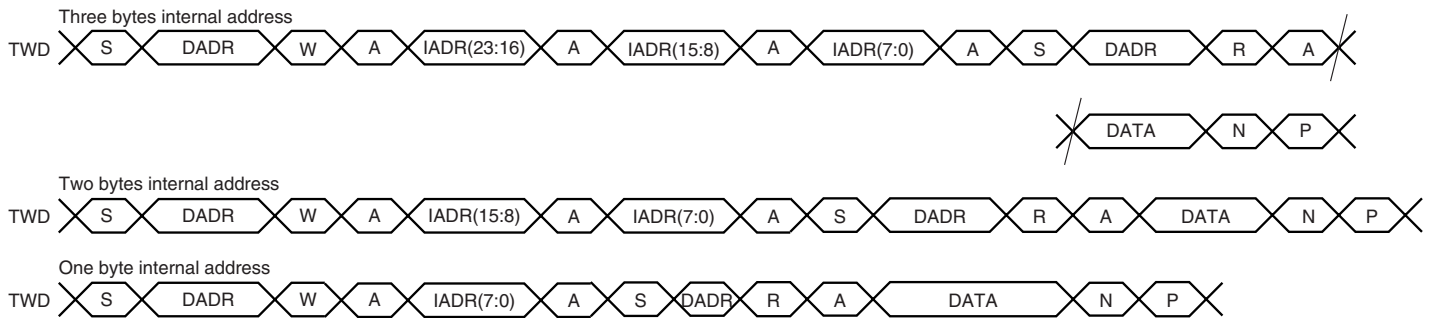
**Figure 93. Master Write with One, Two or Three Bytes Internal Address and One Data Byte**



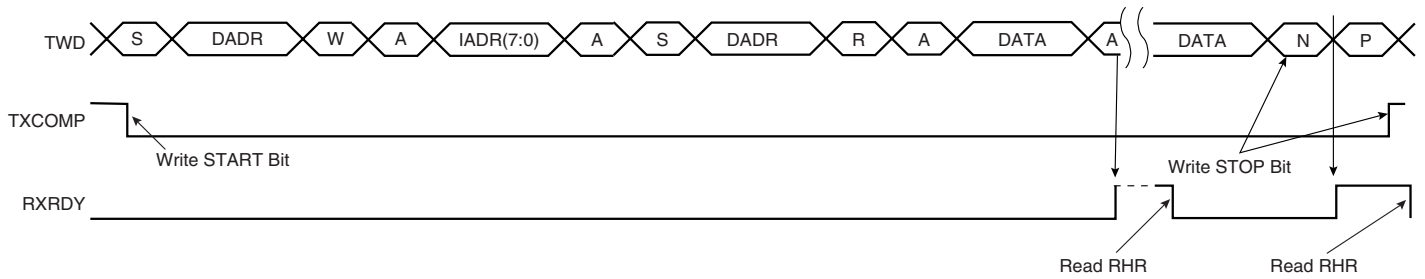
**Figure 94. Master Write with One Byte Internal Address and Multiple Data Bytes**



**Figure 95. Master Read with One, Two or Three Bytes Internal Address and One Data Byte**



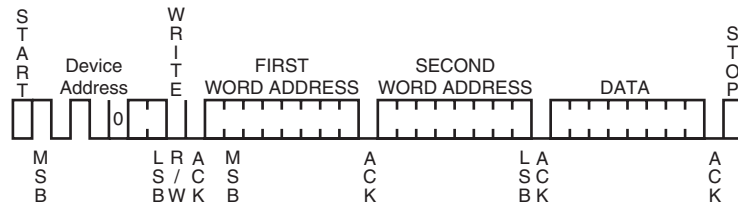
**Figure 96. Master Read with One Byte Internal Address and Multiple Data Bytes**



- S = Start
- P = Stop
- W = Write/Read
- A = Acknowledge
- DADR= Device Address
- IADR = Internal Address

Figure 97 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

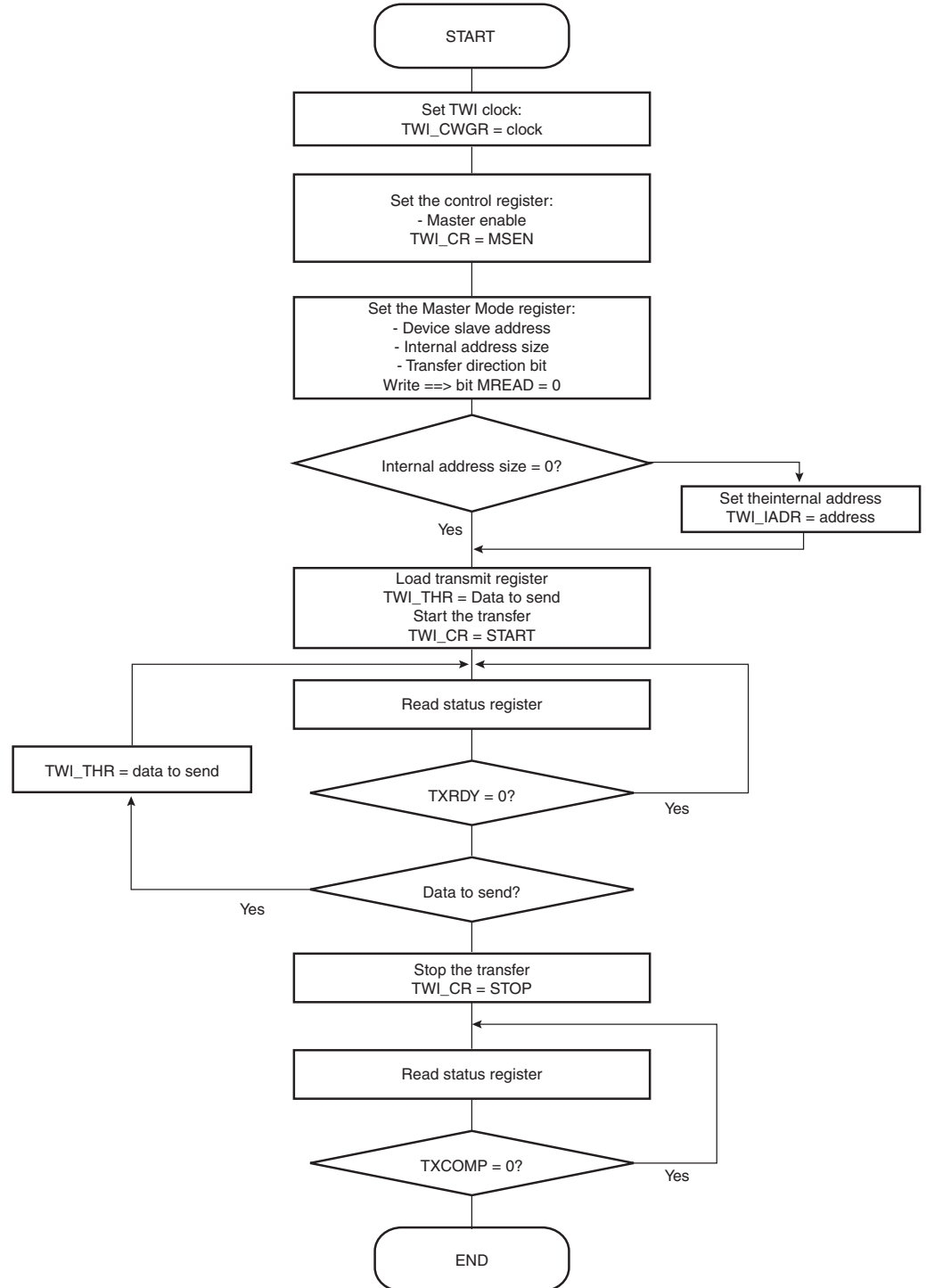
**Figure 97. Internal Address Usage**



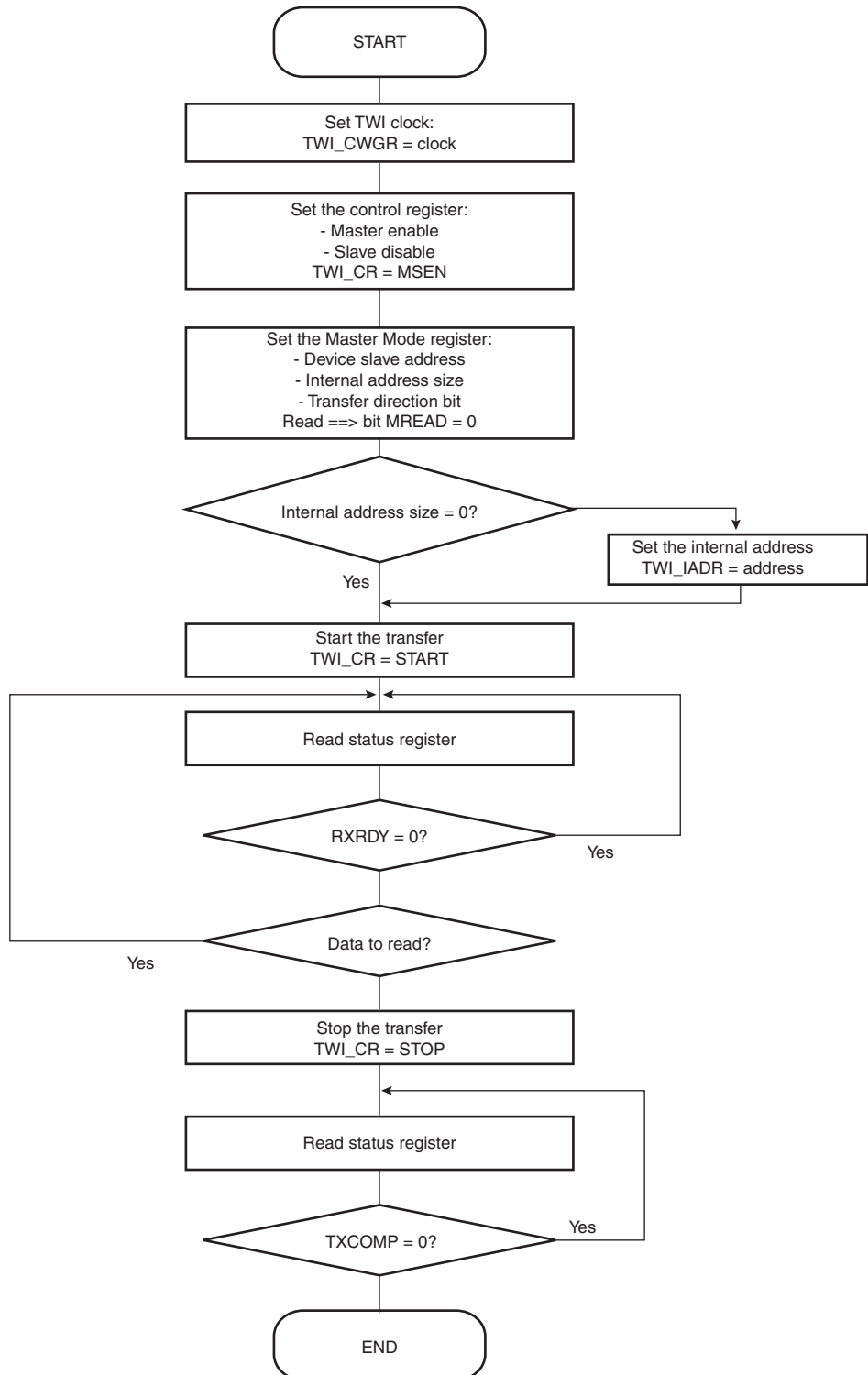
## Read/Write Flowcharts

The following flowcharts shown in Figure 98 on page 254 and in Figure 99 on page 255 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 98.** TWI Write in Master Mode



**Figure 99.** TWI Read in Master Mode



## Two-wire Interface (TWI) User Interface

**Table 57.** 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	–	–	–



## 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.

## 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
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.

## 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.

## 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.

## 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
UNRE	OVRE	–	–	–	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 received (in Slave), 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).

- **OVRE: Overrun Error**

0 = TWI\_RHR has not been loaded while RXRDY was set

1 = TWI\_RHR has been loaded while RXRDY was set. Reset by read in TWI\_SR when TXCOMP is set.

- **UNRE: Underrun Error**

0 = No underrun error

1 = No valid data in TWI\_THR (TXRDY set) while trying to load the data shifter. This action automatically generated a STOP bit in Master Mode. Reset by read in TWI\_SR when TXCOMP is set.

- **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.

## 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
UNRE	OVRE	–	–	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed**
- **RXRDY: Receive Holding Register Ready**
- **TXRDY: Transmit Holding Register Ready**
- **OVRE: Overrun Error**
- **UNRE: Underrun Error**
- **NACK: Not Acknowledge**

0 = No effect.

1 = Enables the corresponding interrupt.



## 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
UNRE	OVRE	–	–	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed**
- **RXRDY: Receive Holding Register Ready**
- **TXRDY: Transmit Holding Register Ready**
- **OVRE: Overrun Error**
- **UNRE: Underrun Error**
- **NACK: Not Acknowledge**

0 = No effect.

1 = Disables the corresponding interrupt.

## 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
UNRE	OVRE	–	–	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed**
- **RXRDY: Receive Holding Register Ready**
- **TXRDY: Transmit Holding Register Ready**
- **OVRE: Overrun Error**
- **UNRE: Underrun Error**
- **NACK: Not Acknowledge**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.



## 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

## 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



## Universal Synchronous/Asynchronous Receiver/Transmitter (USART)

### 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. Multi-drop communications are also supported through address bit handling in reception and transmission.

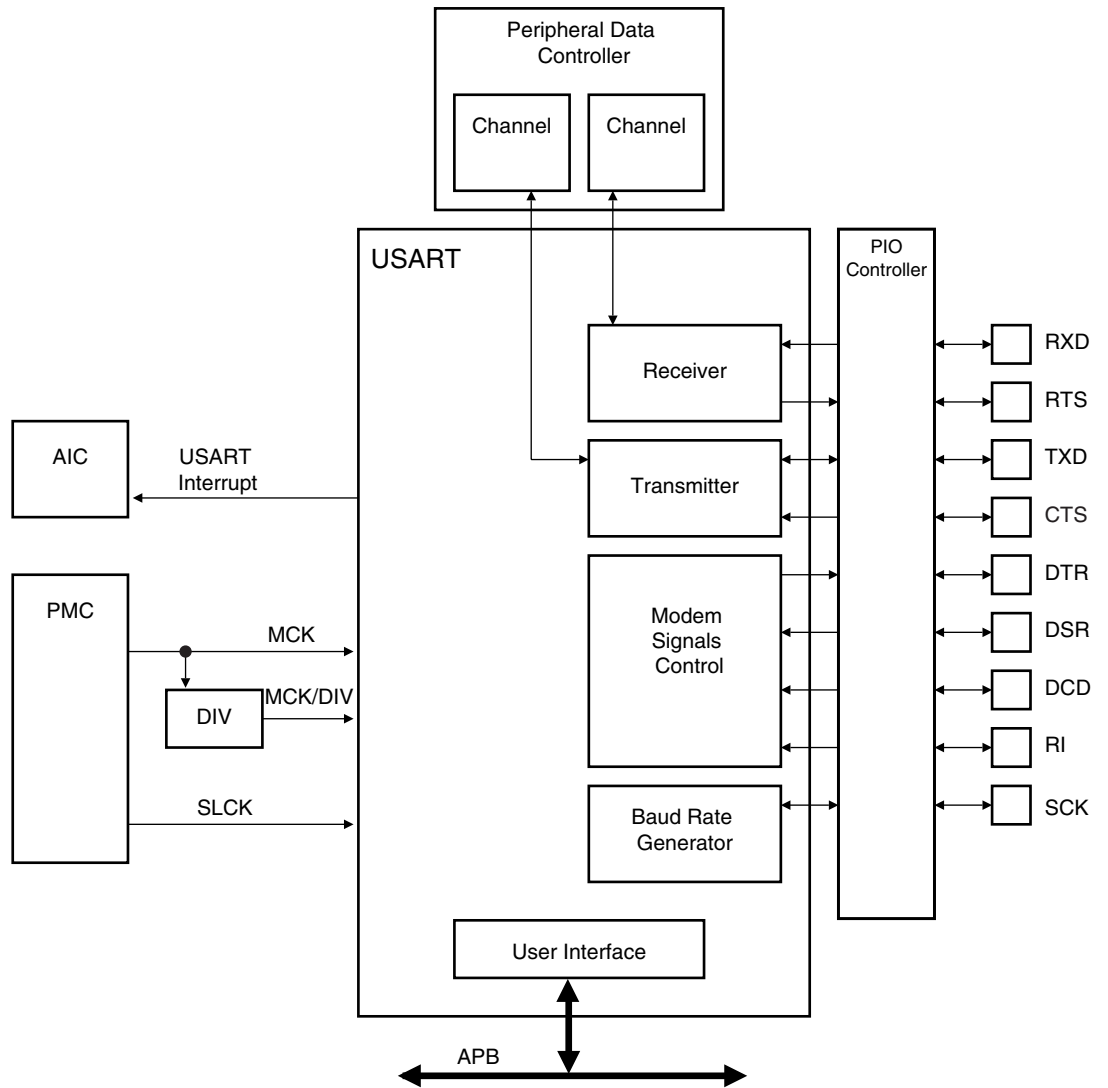
The USART features three test modes: remote loopback, local loopback and automatic echo.

The USART supports specific operating modes providing interfaces on RS485 buses, with ISO7816 T = 0 or T = 1 smart card slots, infrared transceivers and connection to modem ports. 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 Data Controller, which enables data transfers to the transmitter and from the receiver. The PDC provides chained buffer management without any intervention of the processor.

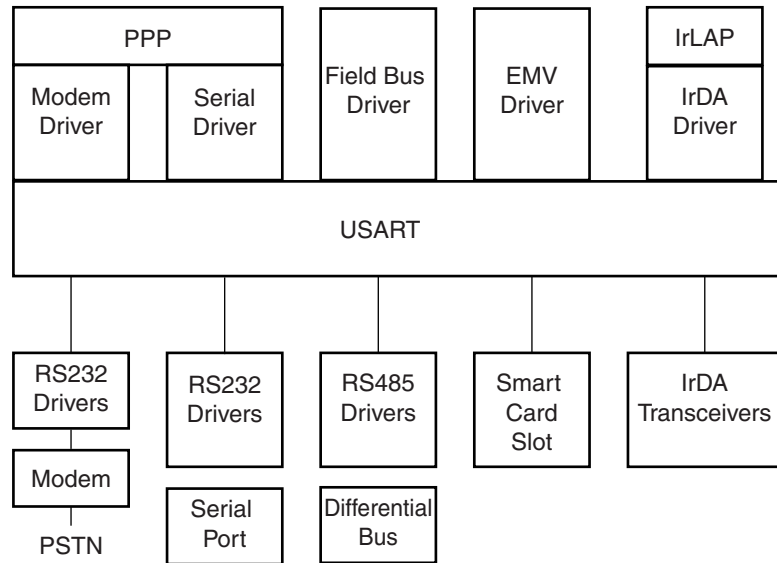
# Block Diagram

Figure 100. USART Block Diagram



## Application Block Diagram

Figure 101. Application Block Diagram



## I/O Lines Description

Table 58. 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	
RI	Ring Indicator	Input	Low
DSR	Data Set Ready	Input	Low
DCD	Data Carrier Detect	Input	Low
DTR	Data Terminal Ready	Output	Low
CTS	Clear to Send	Input	Low
RTS	Request to Send	Output	Low

## Product Dependencies

### 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.

All the pins of the modems may or may not be implemented on the USART within a product.

### 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.

### 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.

## 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 modem signals management
  - Optional break management
  - Optional multi-drop 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 Modem signals management
  - Optional Break management
  - Optional Multi-Drop 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

## Baud Rate Generator

The Baud Rate Generator provides the bit period clock named the Baud Rate Clock to both the receiver and the transmitter.

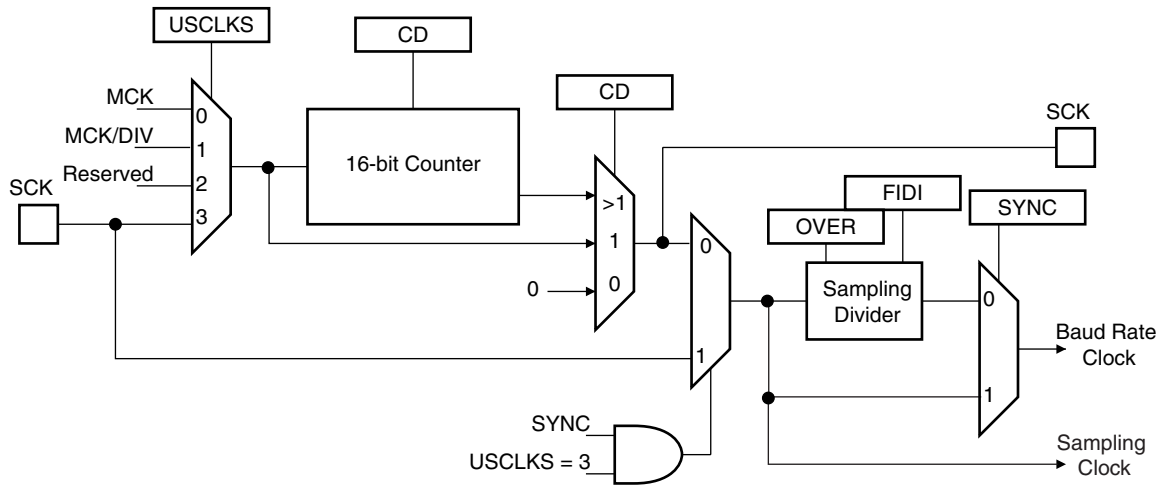
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US\_MR) between:

- 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 102. Baud Rate Generator**



**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.

**Baud Rate Calculation Example**

Table 59 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 59. Baud Rate Example (OVER = 0)**

Source Clock	Expected Baud Rate	Calculation Result	CD	Actual Baud Rate	Error
MHz	Bit/s			Bit/s	
3 686 400	38 400	6.00	6	38 400.00	0.00%
4 915 200	38 400	8.00	8	38 400.00	0.00%
5 000 000	38 400	8.14	8	39 062.50	1.70%
7 372 800	38 400	12.00	12	38 400.00	0.00%
8 000 000	38 400	13.02	13	38 461.54	0.16%
12 000 000	38 400	19.53	20	37 500.00	2.40%
12 288 000	38 400	20.00	20	38 400.00	0.00%
14 318 180	38 400	23.30	23	38 908.10	1.31%
14 745 600	38 400	24.00	24	38 400.00	0.00%

**Table 59.** Baud Rate Example (OVER = 0) (Continued)

Source Clock	Expected Baud Rate	Calculation Result	CD	Actual Baud Rate	Error
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%
60 000 000	38 400	97.66	98	38 265.31	0.35%
70 000 000	38 400	113.93	114	38 377.19	0.06%

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%.

$$Error = 1 - \left( \frac{ExpectedBaudRate}{ActualBaudRate} \right)$$

### 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.

### Baud Rate in ISO 7816 Mode

The ISO7816 specification defines the bit rate with the following formula:

$$B = \frac{Di}{Fi} \times f$$

where:

- B is the bit rate
- Di is the bit-rate adjustment factor
- Fi is the clock frequency division factor
- f is the ISO7816 clock frequency (Hz)

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 60.

**Table 60.** Binary and Decimal Values for D

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 61.

**Table 61.** Binary and Decimal Values for F

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 62 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

**Table 62.** Possible Values for the Fi/Di Ratio

Fi/Di	372	558	774	1116	1488	1806	512	768	1024	1536	2048
1	372	558	744	1116	1488	1860	512	768	1024	1536	2048
2	186	279	372	558	744	930	256	384	512	768	1024
4	93	139.5	186	279	372	465	128	192	256	384	512
8	46.5	69.75	93	139.5	186	232.5	64	96	128	192	256
16	23.25	34.87	46.5	69.75	93	116.2	32	48	64	96	128
32	11.62	17.43	23.25	34.87	46.5	58.13	16	24	32	48	64
12	31	46.5	62	93	124	155	42.66	64	85.33	128	170.6
20	18.6	27.9	37.2	55.8	74.4	93	25.6	38.4	51.2	76.8	102.4

If the USART is configured in ISO7816 Mode, the clock selected by the USCLKS field in the Mode Register (US\_MR) is first divided by the value programmed in the field CD in the Baud Rate Generator Register (US\_BRGR). The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. This means that the CLKO bit can be set in US\_MR.

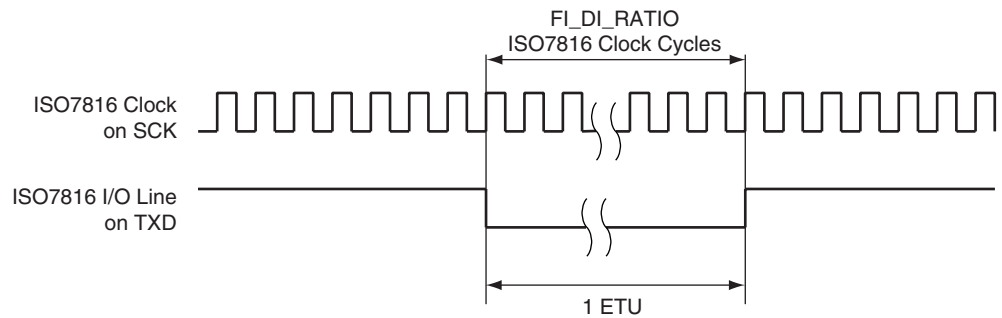
This clock is then divided by the value programmed in the FI\_DI\_RATIO field in the FI\_DI\_Ratio register (US\_FIDI). This is performed by the Sampling Divider, which performs a division by up to 2047 in ISO7816 Mode. The non-integer values of the Fi/Di Ratio are not supported and the user must program the FI\_DI\_RATIO field to a value as close as possible to the expected value.

The FI\_DI\_RATIO field resets to the value 0x174 (372 in decimal) and is the most common divider between the ISO7816 clock and the bit rate (Fi = 372, Di = 1).

Figure 103 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.



**Figure 103.** Elementary Time Unit (ETU)



## 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.

## Synchronous and Asynchronous Modes

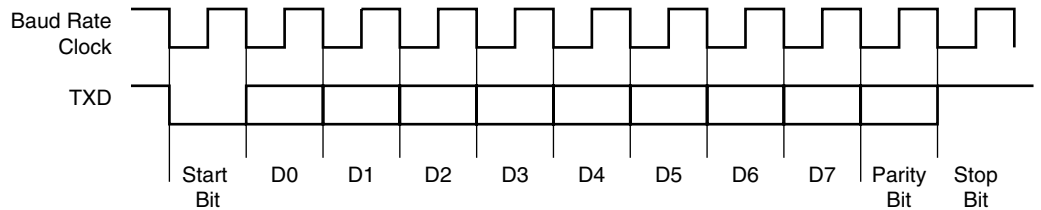
### Transmitter Operations

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 MODE9 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 104.** Character Transmit

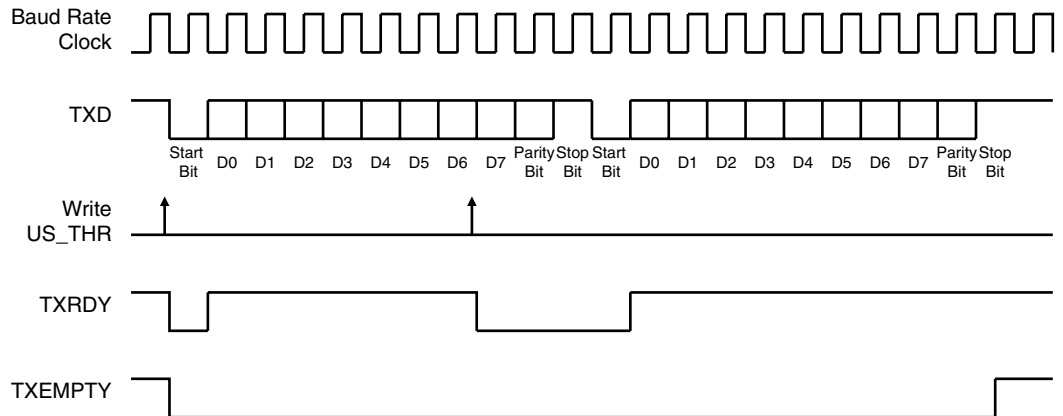
Example: 8-bit, Parity Enabled One Stop



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 105.** Transmitter Status



**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).

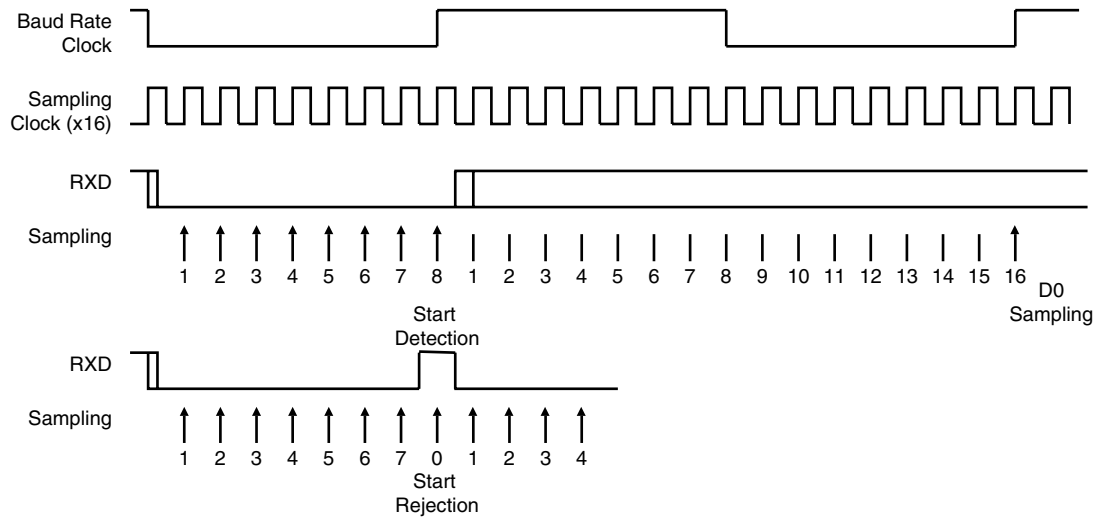
The receiver samples the RXD line. If the line is sampled during one half of a bit time at 0, a start bit is detected and data, parity and stop bits are successively sampled on the bit rate clock.

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. 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.

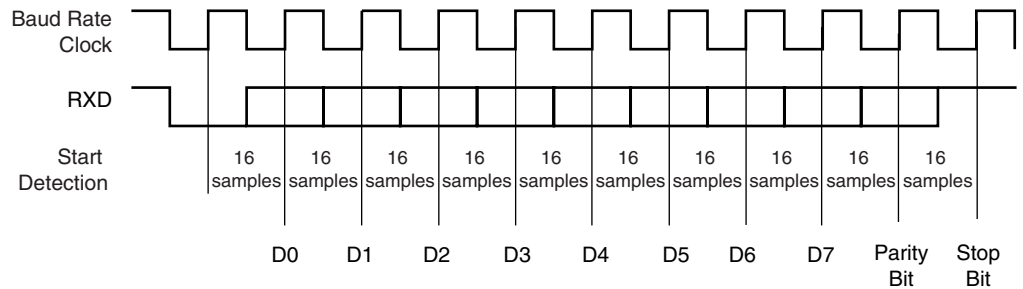
Figure 106 and Figure 107 illustrate start detection and character reception when USART operates in asynchronous mode.

**Figure 106. Asynchronous Start Detection**



**Figure 107. Asynchronous Character Reception**

Example: 8-bit, Parity Enabled



### Synchronous Receiver

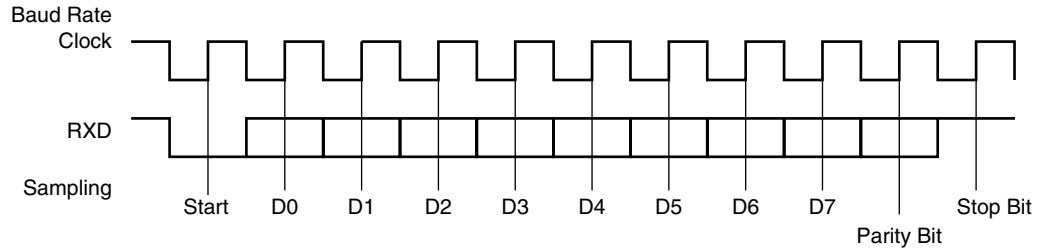
In synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations provide a high speed transfer capability.

Configuration fields and bits are the same as in asynchronous mode.

Figure 108 illustrates a character reception in synchronous mode.

**Figure 108.** Synchronous Mode Character Reception

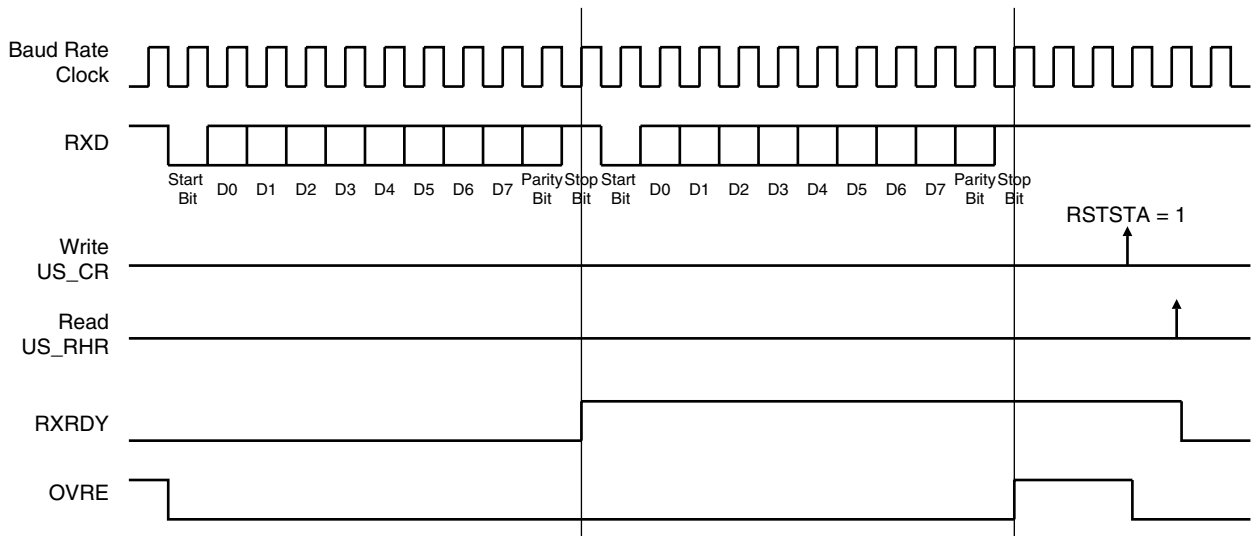
Example: 8-bit, Parity Enabled 1 Stop



### 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 109.** Receiver Status



## 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, which is discussed in a separate paragraph. 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 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 odd 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 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.

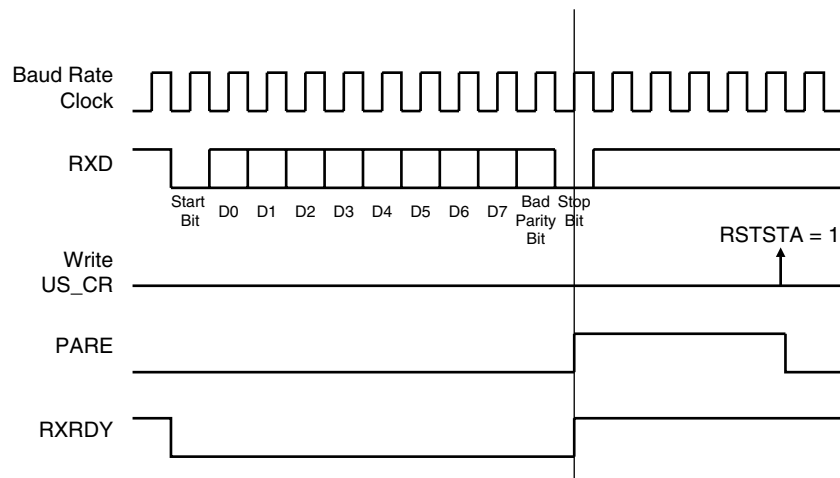
Table 63 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits at 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even. I

**Table 63.** 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 110 illustrates the parity bit status setting and clearing.

**Figure 110.** Parity Error



### Multi-drop Mode

If the PAR field in the Mode Register (US\_MR) is programmed to the value 0x6 or 0x7, the USART runs in Multi-drop 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 multi-drop 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.

### 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 111, 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 111.** Timeguard Operations

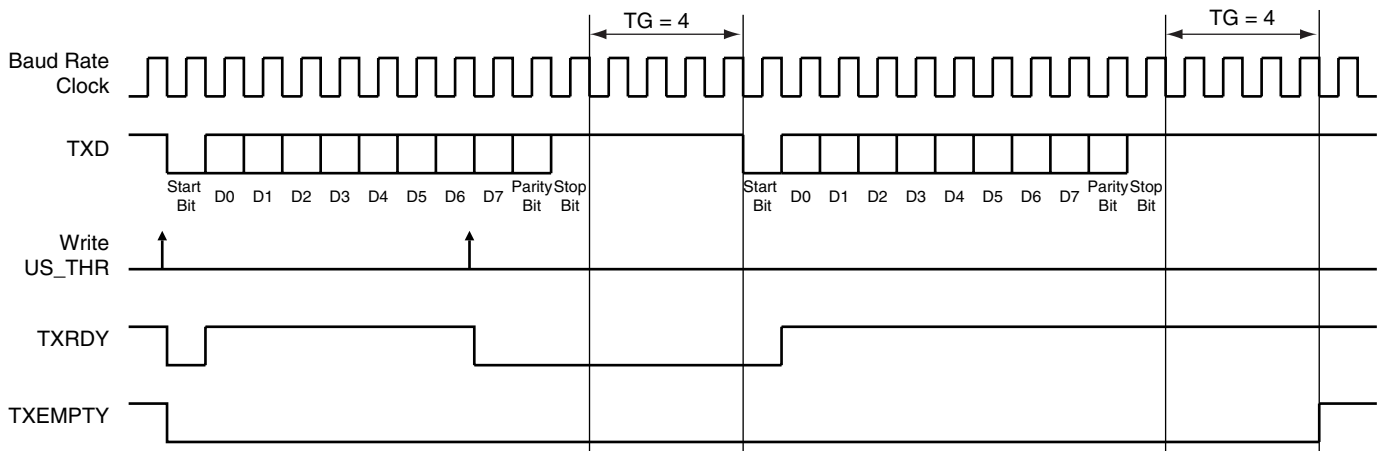


Table 64 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

**Table 64.** Maximum Timeguard Length Depending on Baud Rate

Baud Rate	Bit time	Timeguard
Bit/sec	μs	ms
1 200	833	212.50
9 600	104	26.56
14400	69.4	17.71
19200	52.1	13.28
28800	34.7	8.85
33400	29.9	7.63
56000	17.9	4.55
57600	17.4	4.43
115200	8.7	2.21

## 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:

- Obtain an interrupt when a time-out is detected after having received at least one character. This is performed by writing the Control Register (US\_CR) with the STTTO (Start Time-out) bit at 1.
- Obtain a periodic 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 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 112 shows the block diagram of the Receiver Time-out feature.

**Figure 112.** Receiver Time-out Block Diagram

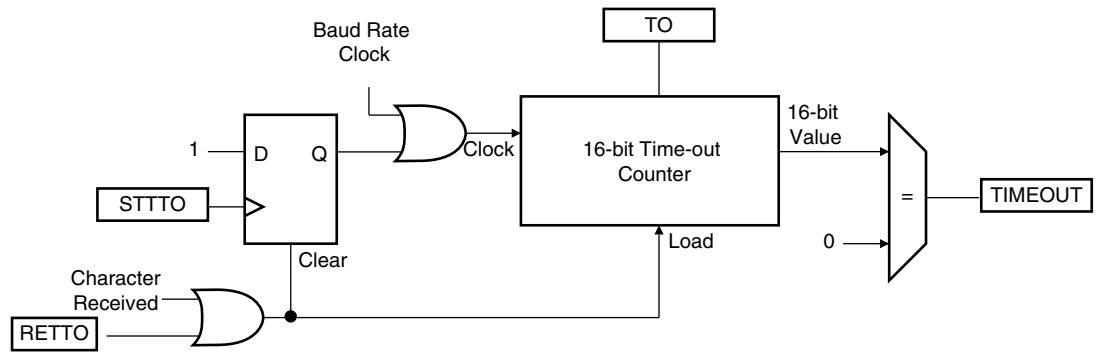


Table 65 gives the maximum time-out period for some standard baud rates.t

**Table 65.** Maximum Time-out Period

Baud Rate	Bit Time	Time-out
bit/sec	µs	ms
600	1 667	109 225
1 200	833	54 613
2 400	417	27 306
4 800	208	13 653
9 600	104	6 827
14400	69	4 551
19200	52	3 413
28800	35	2 276
33400	30	1 962
56000	18	1 170
57600	17	1 138
200000	5	328

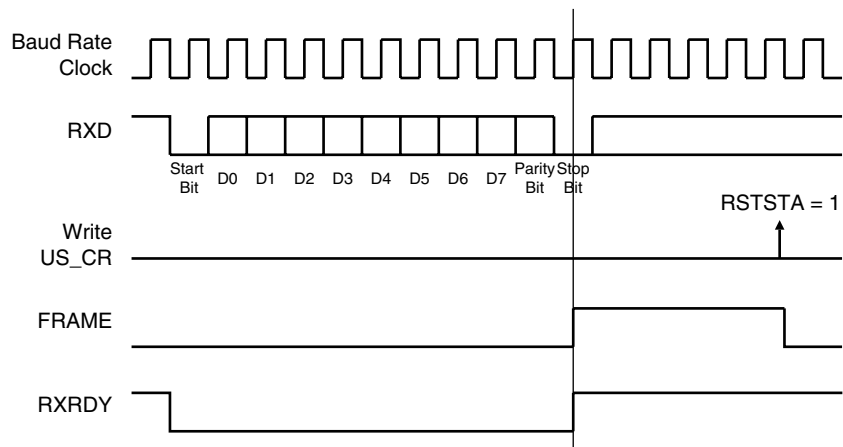
**Framing Error**

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 113. Framing Error Status**



## 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 STTBK and STPBRK 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.

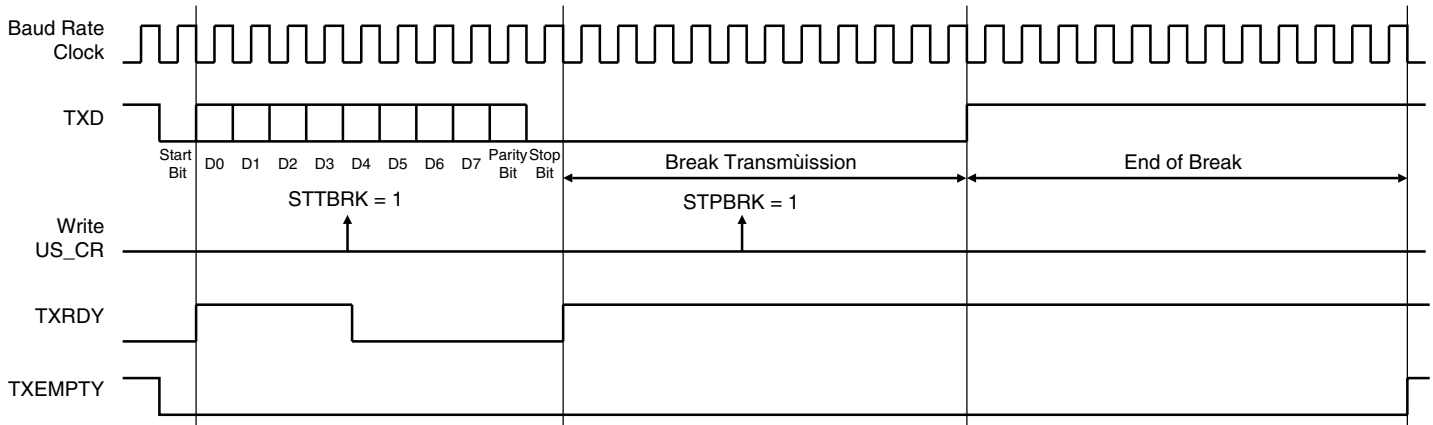
Writing US\_CR with the both STTBK and STPBRK bits at 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBK command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

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 114 illustrates the effect of both the Start Break (STTBK) and Stop Break (STP BRK) commands on the TXD line.

**Figure 114. Break Transmission**



**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.

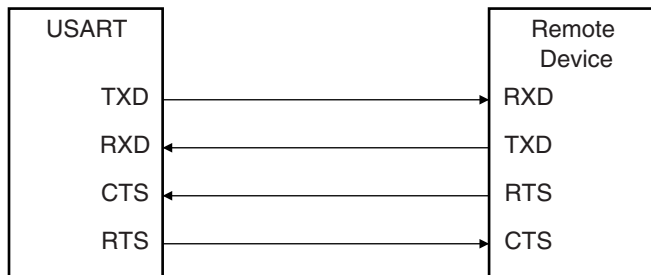
When the low stop bit is detected, the receiver asserts the RXBRK bit in US\_CSR. This bit may be cleared by writing the Control Register (US\_CR) with the bit RSTSTA at 1.

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.

**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 115.

**Figure 115. 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 116 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 116.** Receiver Behavior when Operating with Hardware Handshaking

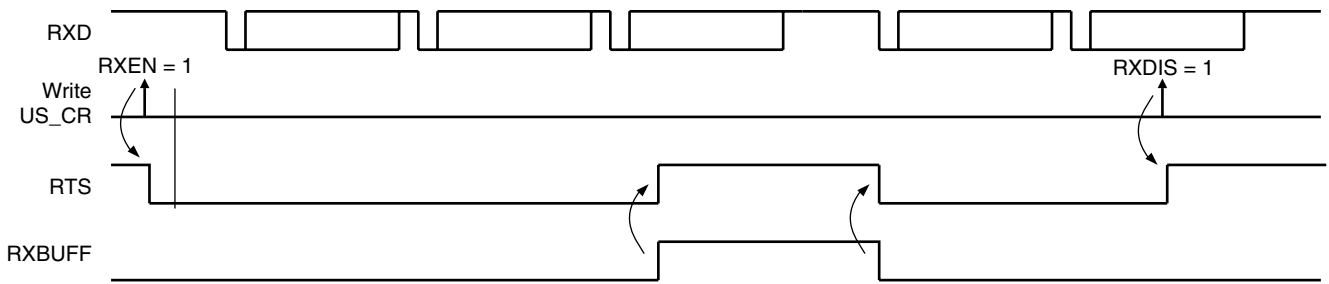


Figure 117 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

**Figure 117.** Transmitter Behavior when Operating with Hardware Handshaking



## 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.

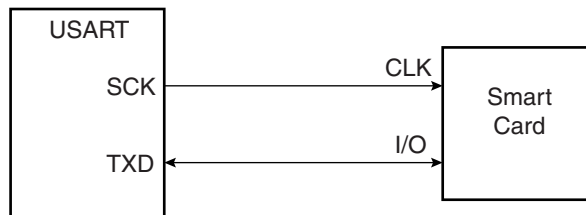
Setting the USART in ISO7816 mode is performed by writing the USART\_MODE field in the Mode Register (US\_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

## 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 269).

The USART connects to a smart card as shown in Figure 118. 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 118.** 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.

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).

**Protocol T = 0**

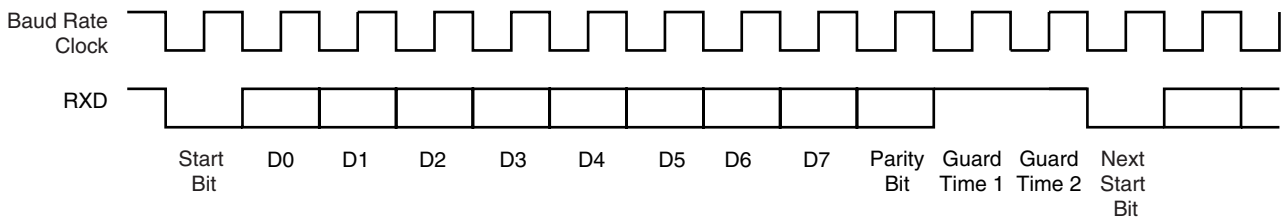
In T = 0 protocol, a character is made up of one start bit, eight data bits, one parity bit and one guard time, which lasts two bit times. The transmitter shifts out the bits and does not drive the I/O line during the guard time.

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 119.

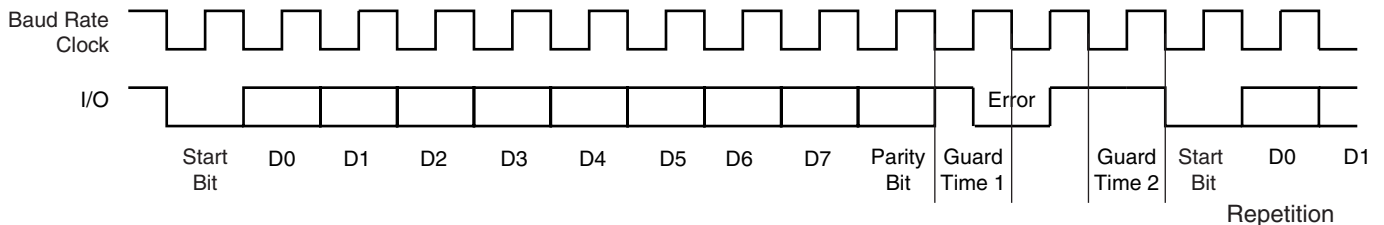
If a parity error is detected by the receiver, it drives the I/O line at 0 during the guard time, as shown in Figure 120. This error bit is also named NACK, for Non Acknowledge. In this case, the character lasts 1 bit time more, as the guard time length is the same and is added to the error bit time 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 119.** T = 0 Protocol without Parity Error



**Figure 120.** T = 0 Protocol with Parity Error



*Receive Error Counter*

The USART receiver also records the total number of errors. This can be read in the Number of Error (US\_NER) register. The NB\_ERRORS field can record up to 255 errors. Reading US\_NER automatically clears the NB\_ERRORS field.

*Receive NACK Inhibit*

The USART can also be configured to inhibit an error. This can be achieved by setting the INACK bit in the Mode Register (US\_MR). If INACK is 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.

## 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.

## 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.

## Protocol T = 1

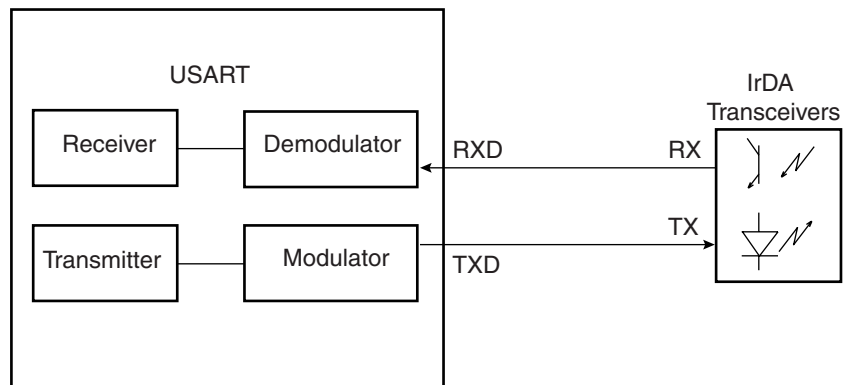
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US\_CSR).

## 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 121. 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 121.** Connection to IrDA Transceivers



The receiver and the transmitter must be enabled or disabled according to the direction of the transmission to be managed.

### IrDA Modulation

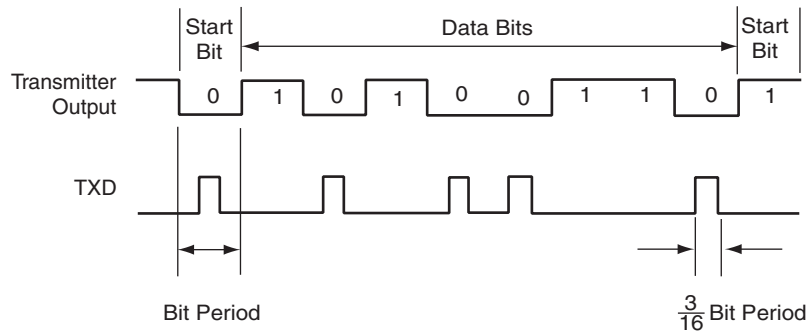
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. "0" is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 66.

**Table 66.** 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 122 shows an example of character transmission.

**Figure 122.** IrDA Modulation



### IrDA Baud Rate

Table 67 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 67.** 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 67.** IrDA Baud Rate Error (Continued)

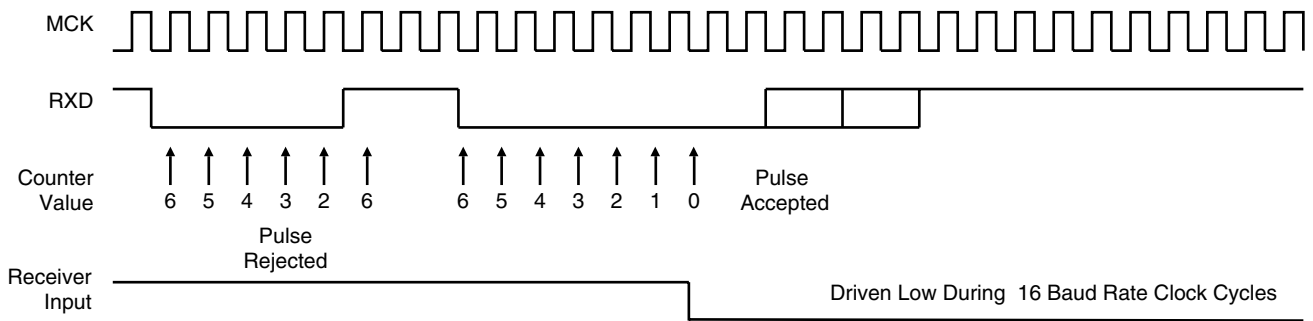
Peripheral Clock	Baud Rate	CD	Baud Rate Error	Pulse Time
32 768 000	38 400	53	0.63%	4.88
40 000 000	38 400	65	0.16%	4.88
3 686 400	19 200	12	0.00%	9.77
20 000 000	19 200	65	0.16%	9.77
32 768 000	19 200	107	0.31%	9.77
40 000 000	19 200	130	0.16%	9.77
3 686 400	9 600	24	0.00%	19.53
20 000 000	9 600	130	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

## 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 123 illustrates the operations of the IrDA demodulator.

**Figure 123.** IrDA Demodulator Operations

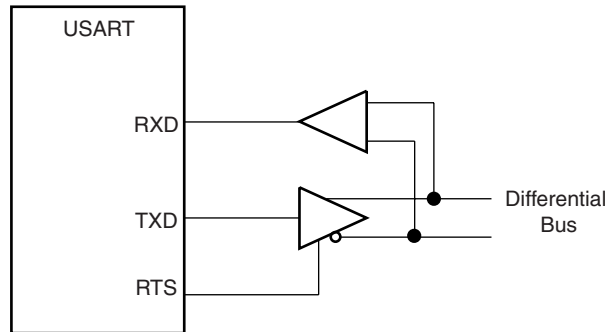


As the IrDA mode uses the same logic as the ISO7816, note that the FI\_DI\_RATIO field in US\_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

## 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 124.

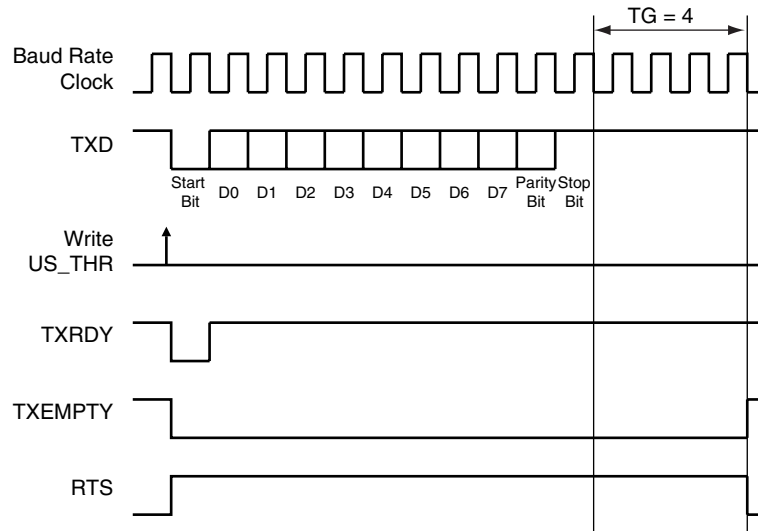
**Figure 124.** Typical Connection to a RS485 Bus



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 125 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

**Figure 125.** Example of RTS Drive with Timeguard





## Modem Mode

The USART features modem mode, which enables control of the signals: DTR (Data Terminal Ready), DSR (Data Set Ready), RTS (Request to Send), CTS (Clear to Send), DCD (Data Carrier Detect) and RI (Ring Indicator). While operating in modem mode, the USART behaves as a DTE (Data Terminal Equipment) as it drives DTR and RTS and can detect level change on DSR, DCD, CTS and RI.

Setting the USART in modem mode is performed by writing the USART\_MODE field in the Mode Register (US\_MR) to the value 0x3. While operating in modem mode the USART behaves as though in asynchronous mode and all the parameter configurations are available.

Table 68 gives the correspondence of the USART signals with modem connection standards.

**Table 68.** Circuit References

USART Pin	V24	CCITT	Direction
TXD	2	103	From terminal to modem
RTS	4	105	From terminal to modem
DTR	20	108.2	From terminal to modem
RXD	3	104	From modem to terminal
CTS	5	106	From terminal to modem
DSR	6	107	From terminal to modem
DCD	8	109	From terminal to modem
RI	22	125	From terminal to modem

The control of the RTS and DTR output pins is performed by writing the Control Register (US\_CR) with the RTSDIS, RTSEN, DTRDIS and DTREN bits respectively at 1. The disable command forces the corresponding pin to its inactive level, i.e. high. The enable commands force the corresponding pin to its active level, i.e. low.

The level changes are detected on the RI, DSR, DCD and CTS pins. If an input change is detected, the RIIC, DSRIC, DCDIC and CTSIC bits in the Channel Status Register (US\_CSR) are set respectively and can trigger an interrupt. The status is automatically cleared when US\_CSR is read. Furthermore, the CTS automatically disables the transmitter when it is detected at its inactive state. If a character is being transmitted when the CTS rises, the character transmission is completed before the transmitter is actually disabled.

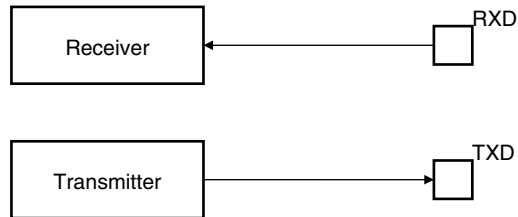
## Test Modes

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.

### Normal Mode

Normal mode connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

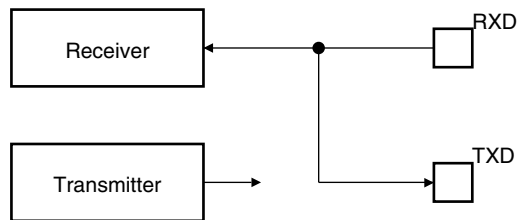
**Figure 126.** Normal Mode Configuration



### 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 127. 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.

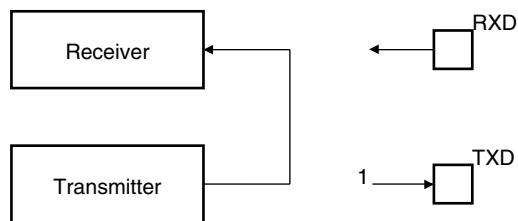
**Figure 127.** Automatic Echo Mode Configuration



### Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 128. 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.

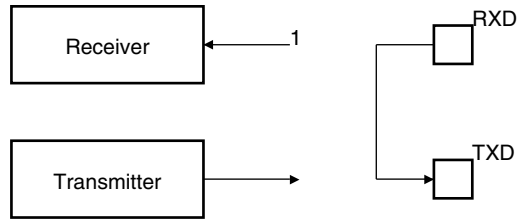
**Figure 128.** Local Loopback Mode Configuration



### Remote Loopback Mode

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 129. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 129. Remote Loopback Mode Configuration



## USART User Interface

**Table 69.** USART Register Mapping

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	0
0x0014	Channel Status Register	US_CSR	Read-only	–
0x0018	Receiver Holding Register	US_RHR	Read-only	0
0x001C	Transmitter Holding Register	US_THR	Write-only	–
0x0020	Baud Rate Generator Register	US_BRGR	Read/Write	0
0x0024	Receiver Time-out Register	US_RTOR	Read/Write	0
0x0028	Transmitter Timeguard Register	US_TTGR	Read/Write	0
0x002C - 0x003C	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	0
0x0050 - 0x00FC	Reserved	–	–	–
0x0100 - 0x0128	Reserved for PDC Registers	–	–	–

## 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	DTRDIS	DTREN
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 the 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.

- **SENDA: Send Address**

0: No effect.

1: In Multi-drop 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

- **DTREN: Data Terminal Ready Enable**

0: No effect.

1: Drives the pin DTR at 0.

- **DTRDIS: Data Terminal Ready Disable**

0: No effect.

1: Drives the pin DTR to 1.

- **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.

## 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	Modem
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	Multi-drop 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.

- **CKLO: 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).



## 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	DCDIC	DSRIC	RIIC
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**
- **RIIC: Ring Indicator Input Change Enable**
- **DSRIC: Data Set Ready Input Change Enable**
- **DCDIC: Data Carrier Detect Input Change Interrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

## 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	DCDIC	DSRIC	RIIC
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**
- **RIIC: Ring Indicator Input Change Disable**
- **DSRIC: Data Set Ready Input Change Disable**
- **DCDIC: Data Carrier Detect Input Change Interrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.



## 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	DCDIC	DSRIC	RIIC
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**
- **RIIC: Ring Indicator Input Change Mask**
- **DSRIC: Data Set Ready Input Change Mask**
- **DCDIC: Data Carrier Detect Input Change Interrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

## 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	DCD	DSR	RI	CTSIC	DCDIC	DSRIC	RIIC
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 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 or the Time-out Register is 0.

1: There has been a time-out since the last Start Time-out command.

- **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.

- **RIIC: Ring Indicator Input Change Flag**

0: No input change has been detected on the RI pin since the last read of US\_CSR.

1: At least one input change has been detected on the RI pin since the last read of US\_CSR.

- **DSRIC: Data Set Ready Input Change Flag**

0: No input change has been detected on the DSR pin since the last read of US\_CSR.

1: At least one input change has been detected on the DSR pin since the last read of US\_CSR.

- **DCDIC: Data Carrier Detect Input Change Flag**

0: No input change has been detected on the DCD pin since the last read of US\_CSR.

1: At least one input change has been detected on the DCD pin since the last read of US\_CSR.

- **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.

- **RI: Image of RI Input**

0: RI is at 0.

1: RI is at 1.

- **DSR: Image of DSR Input**

0: DSR is at 0

1: DSR is at 1.

- **DCD: Image of DCD Input**

0: DCD is at 0.

1: DCD is at 1.

- **CTS: Image of CTS Input**

0: CTS is at 0.

1: CTS is at 1.

## 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
–	–	–	–	–	–	–	RXCHR
7	6	5	4	3	2	1	0
RXCHR							

- **RXCHR: Received Character**

Last character received if RXRDY is set.

## 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
–	–	–	–	–	–	–	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.



## 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



## 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.



## 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.

## 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.



## 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.

## 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.

## Synchronous Serial Controller (SSC)

### 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. Transfers contain up to 16 data of up to 32 bits. They can be programmed to start automatically or on different events detected on the Frame Sync signal.

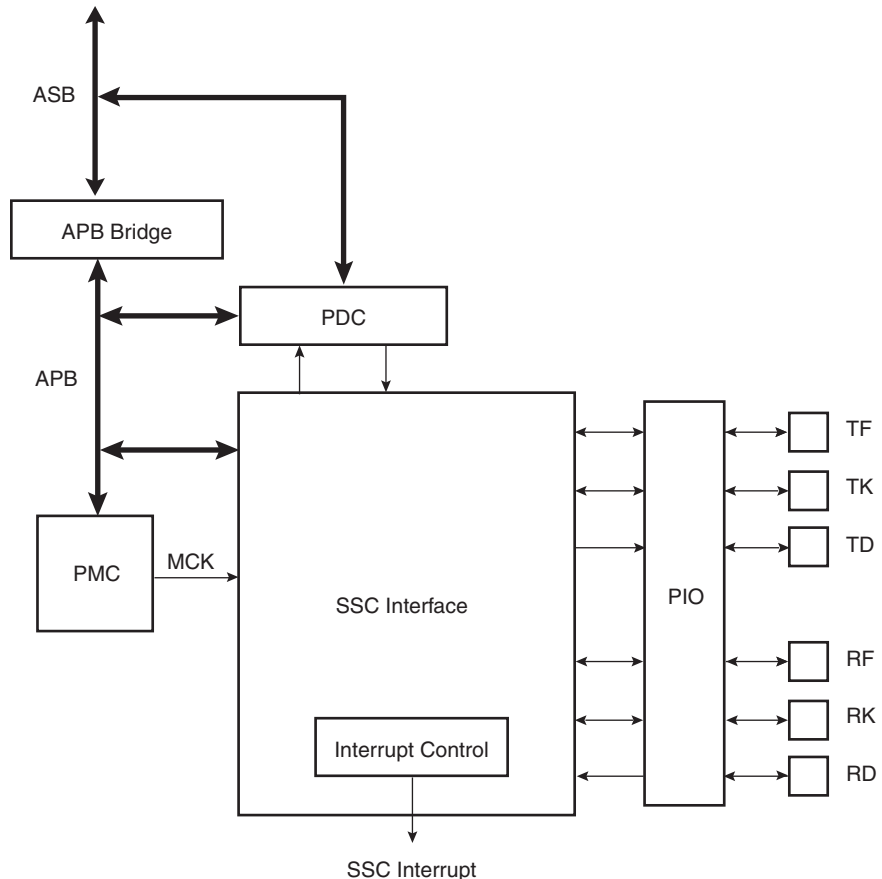
The SSC's high-level of programmability and its two dedicated PDC channels of up to 32 bits permit a continuous high bit rate data transfer without processor intervention.

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

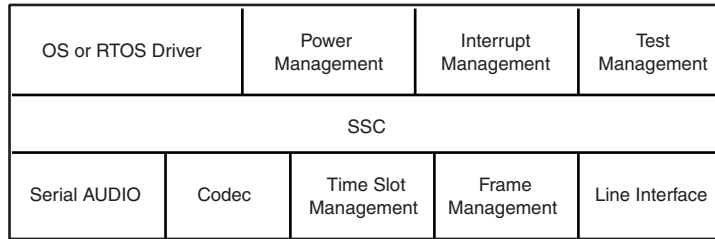
### Block Diagram

Figure 130. Block Diagram



# Application Block Diagram

Figure 131. Application Block Diagram



## Pin Name List

**Table 70.** 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

## Product Dependencies

### 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.

### 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.

### 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.

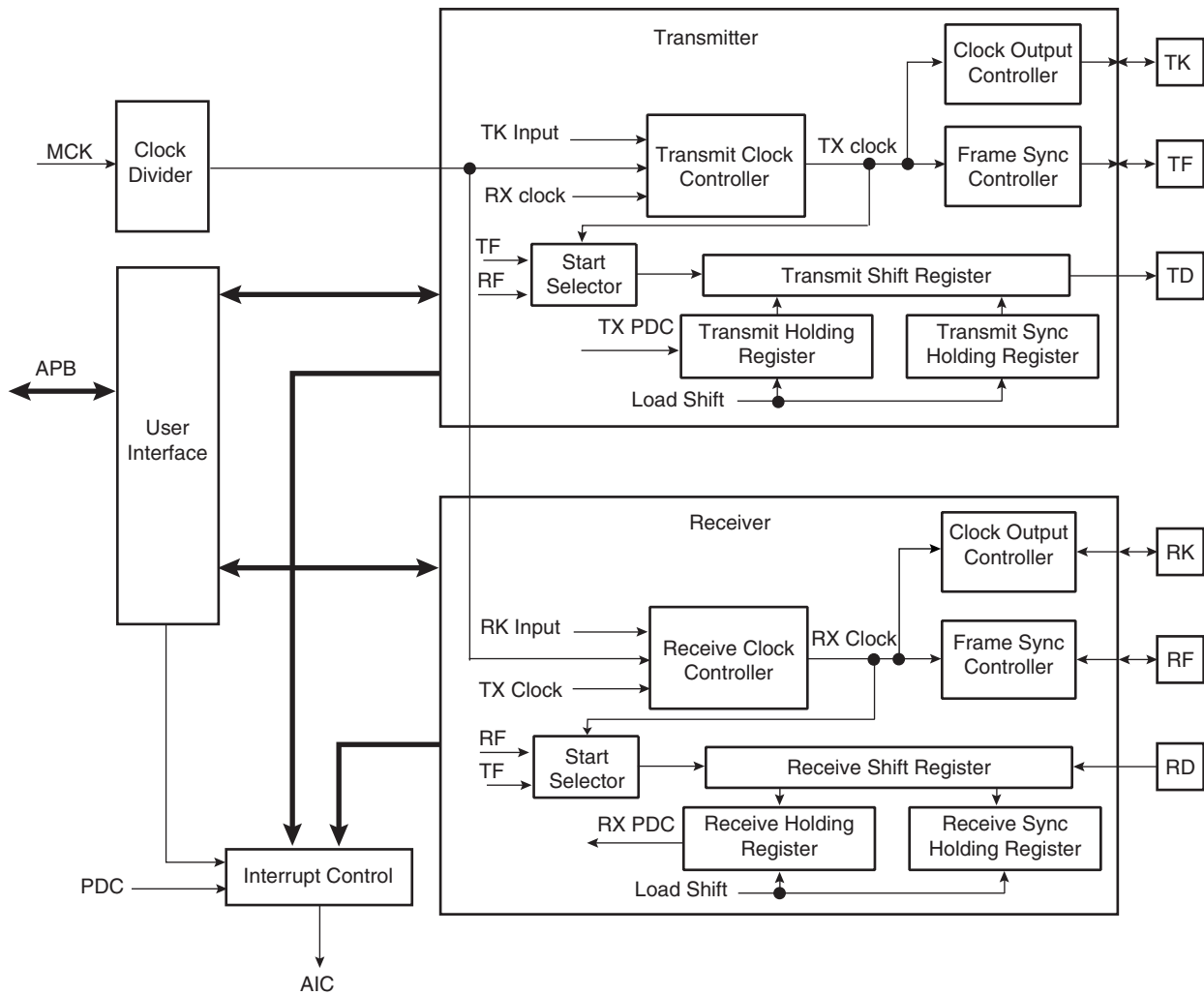
All SSC interrupts can be enabled/disabled configuring the SSC Interrupt mask register. Each pending and unmasked SSC interrupt will assert the SSC interrupt line. The SSC interrupt service routine can get the interrupt origin by reading the SSC interrupt status register.

## 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. Each level of the clock must be stable for at least two master clock periods.

**Figure 132. SSC Functional Block Diagram**



## 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

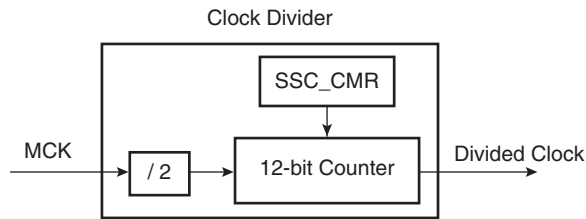
Furthermore, the transmitter block can generate an external clock on the TK I/O pad, and the receiver block can generate an external clock on the RK I/O pad.

This allows the SSC to support many Master and Slave-mode data transfers.



## Clock Divider

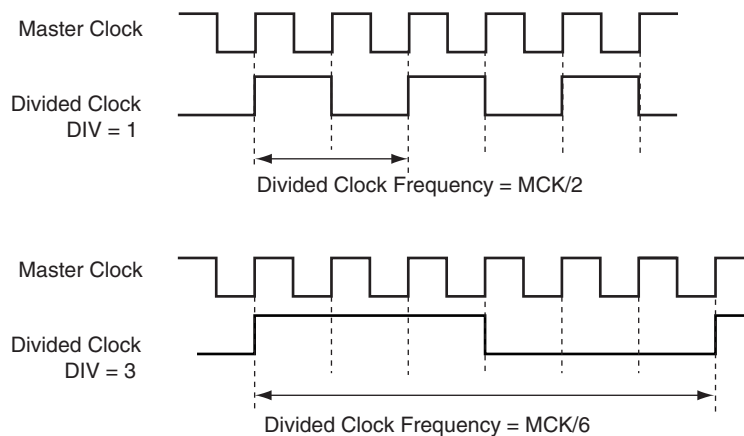
**Figure 133.** Divided Clock Block diagram



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 or greater to 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 if the DIV value is even or odd.

**Figure 134.** Divided Clock Generation



**Table 71.** Bit Rate

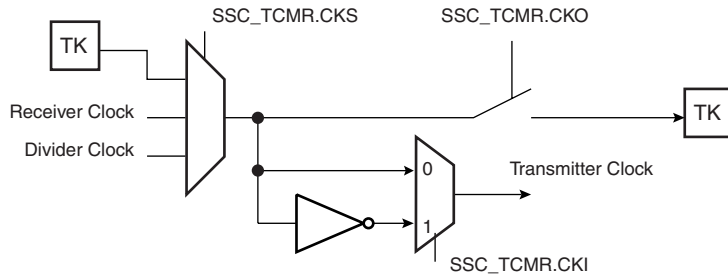
Maximum	Minimum
MCK / 2	MCK / 8190

## 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 Inversion (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 135. Transmitter Clock Management**

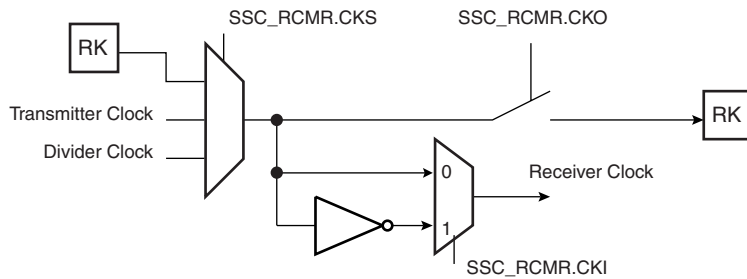


**Receiver Clock Management**

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) might lead to unpredictable results.

**Figure 136. Receiver Clock Management**



## Transmitter Operations

A transmitted frame is triggered by a start event and can be followed by synchronization data before data transmission.

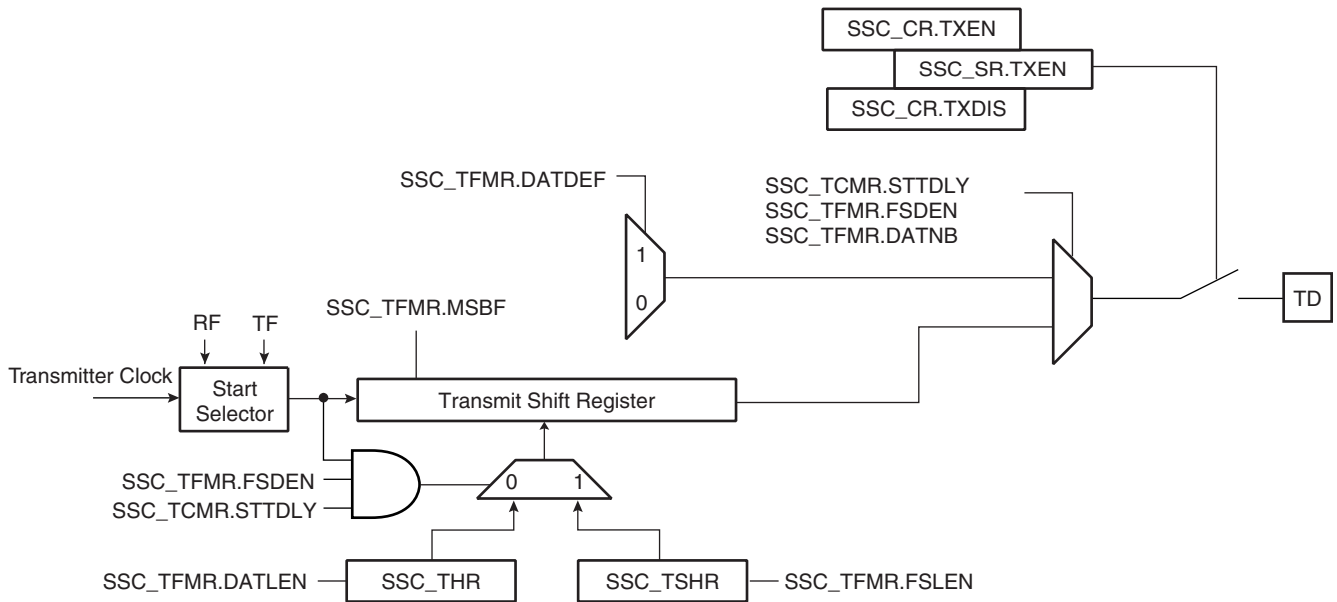
The start event is configured by setting the Transmit Clock Mode Register (SSC\_TCMR). See “Start” on page 316.

The frame synchronization is configured setting the Transmit Frame Mode Register (SSC\_TFMR). See “Frame Sync” on page 318.

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 137.** Transmitter Block Diagram



## Receiver Operations

A received frame is triggered by a start event and can be followed by synchronization data before data transmission.

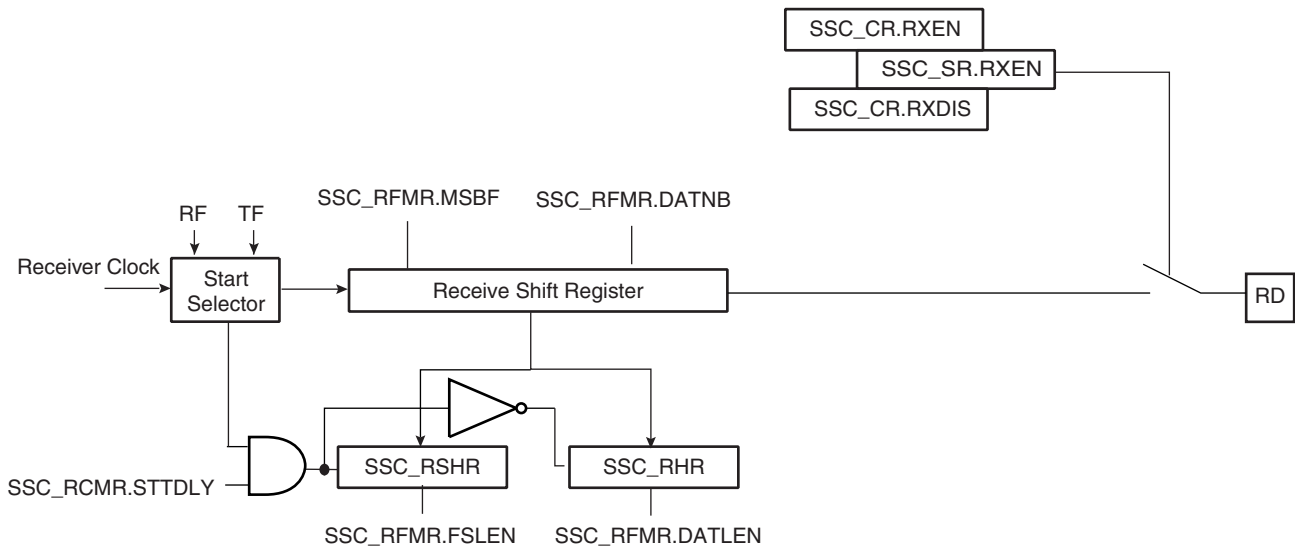
The start event is configured setting the Receive Clock Mode Register (SSC\_RCMR). See “Start” on page 316.

The frame synchronization is configured setting the Receive Frame Mode Register (SSC\_RFMR). See “Frame Sync” on page 318.

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 in function of 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 the RHR register, the status flag OVERUN is set in SSC\_SR and the receiver shift register is transferred in the RHR register.

**Figure 138.** Receiver Block Diagram



## 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:

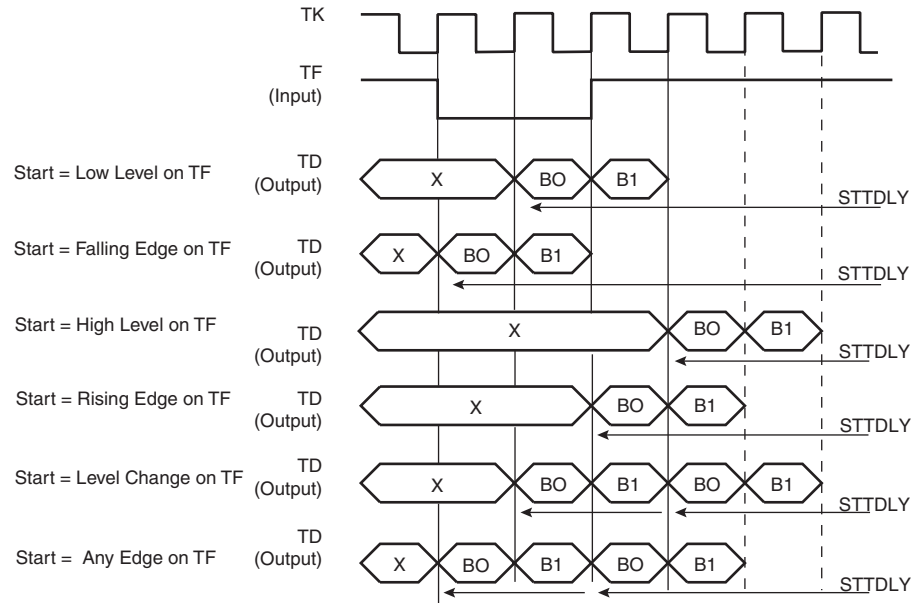
- Continuous. In this case, the transmission starts as soon as a word is written in SSC\_THR and the reception starts as soon as the Receiver is enabled.
- Synchronously with the transmitter/receiver
- On detection of a falling/rising edge on TK/RK
- On detection of a low level/high level on TK/RK
- On detection of a level change or an edge on TK/RK

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).

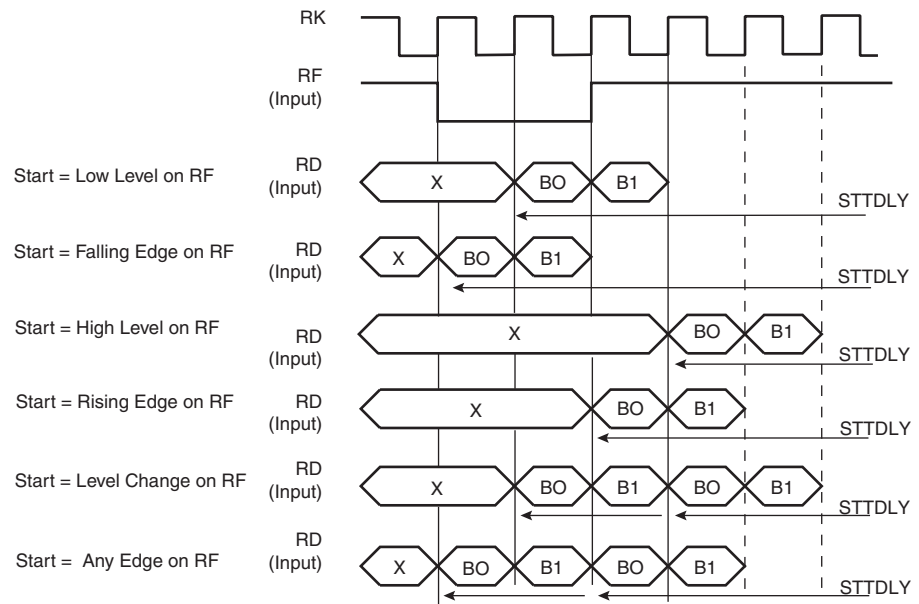
Detection on TF/RF input/output is done through the field FSOS of the Transmit/Receive Frame Mode Register (TFMR/RFMR).

Generating a Frame Sync signal is not possible without generating it on its related output.

**Figure 139. Transmit Start Mode**



**Figure 140. Receive Pulse/Edge Start Modes**



## 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.

## Frame Sync Data

Frame Sync Data transmits or receives a specific tag during the Frame Synchro 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.

## 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).

## Data Format

The data framing format of both the transmitter and the receiver are largely 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:

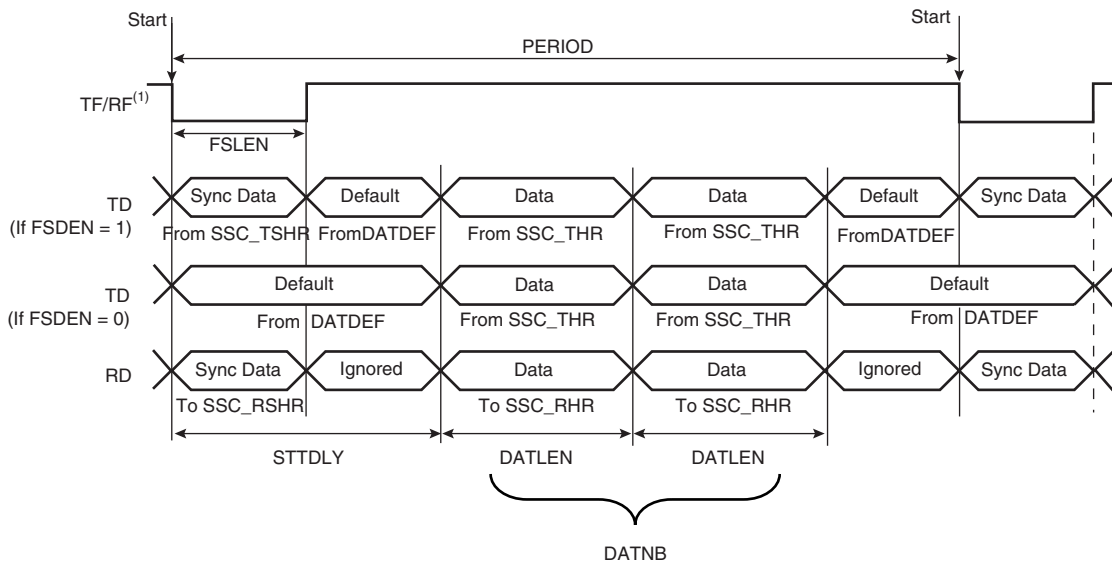
- The event that starts the data transfer (START).
- The delay in number of bit periods between the start event and the first data bit (STTDLY).
- The length of the data (DATLEN)
- The number of data to be transferred for each start event (DATNB).
- The length of Synchronization transferred for each 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 72.** Data Frame Registers

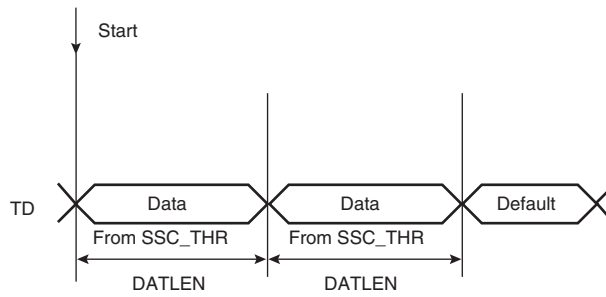
Transmitter	Receiver	Field	Length	Comment
SSC_TFMR	SSC_RFMR	DATLEN	Up to 32	Size of word
SSC_TFMR	SSC_RFMR	DATNB	Up to 16	Number Word transmitter in frame
SSC_TFMR	SSC_RFMR	MSBF		1 most significant bit in first
SSC_TFMR	SSC_RFMR	FSLEN	Up to 16	Size of Synchro data register
SSC_TFMR		DATDEF	0 or 1	Data 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 141.** Transmit and Receive Frame Format in Edge/Pulse Start Modes



Note: 1. Example of Input on falling edge of TF/RF.

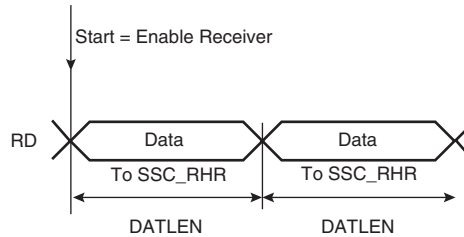
**Figure 142.** Transmit Frame Format in Continuous Mode



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 143.** Receive Frame Format in Continuous Mode



Note: 1. STTDLY is set to 0.

## 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.

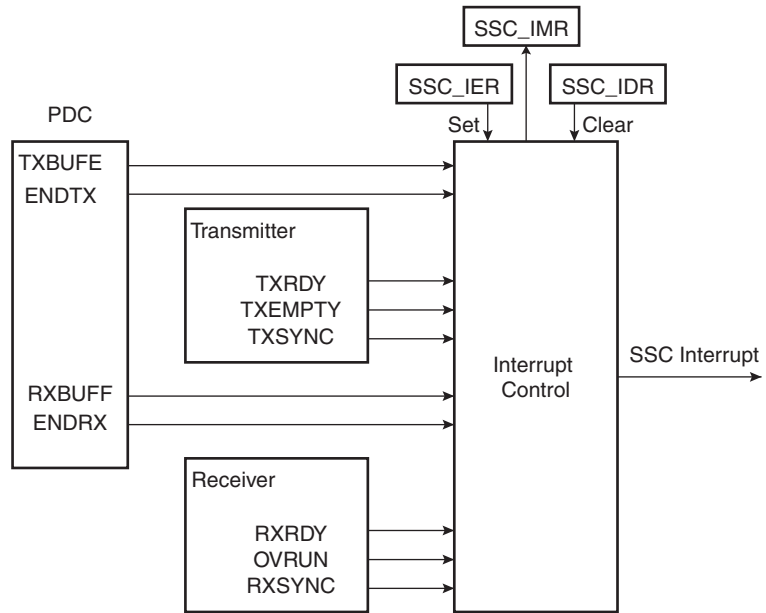
## Interrupt

Most bits in SSC\_SR have a corresponding bit in interrupt management registers.

The SSC Controller 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), which respectively enable and disable 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.



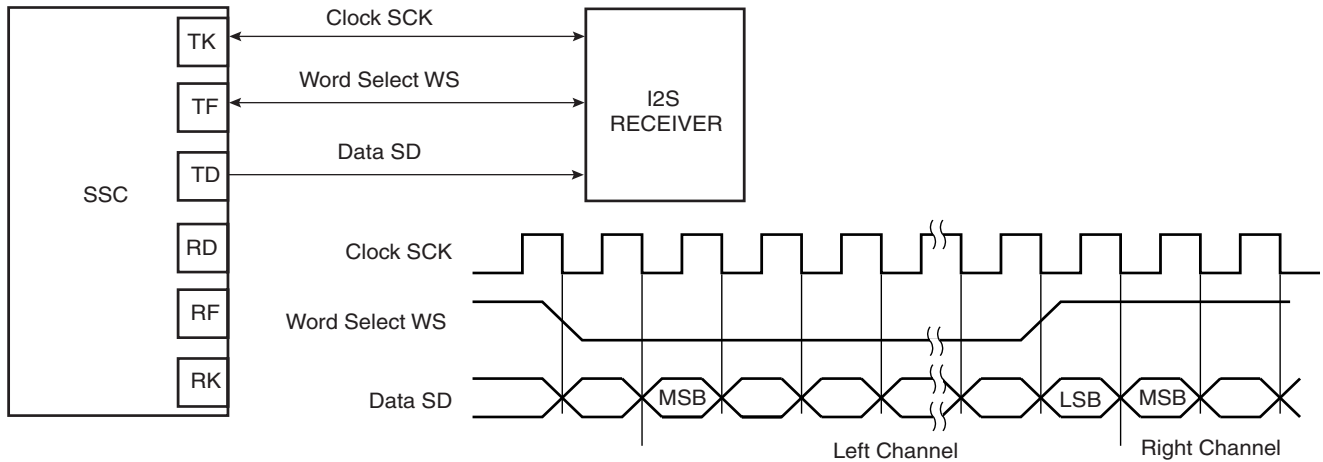
Figure 144. Interrupt Block Diagram



## 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 145.** Audio Application Block Diagram



**Figure 146.** Codec Application Block Diagram

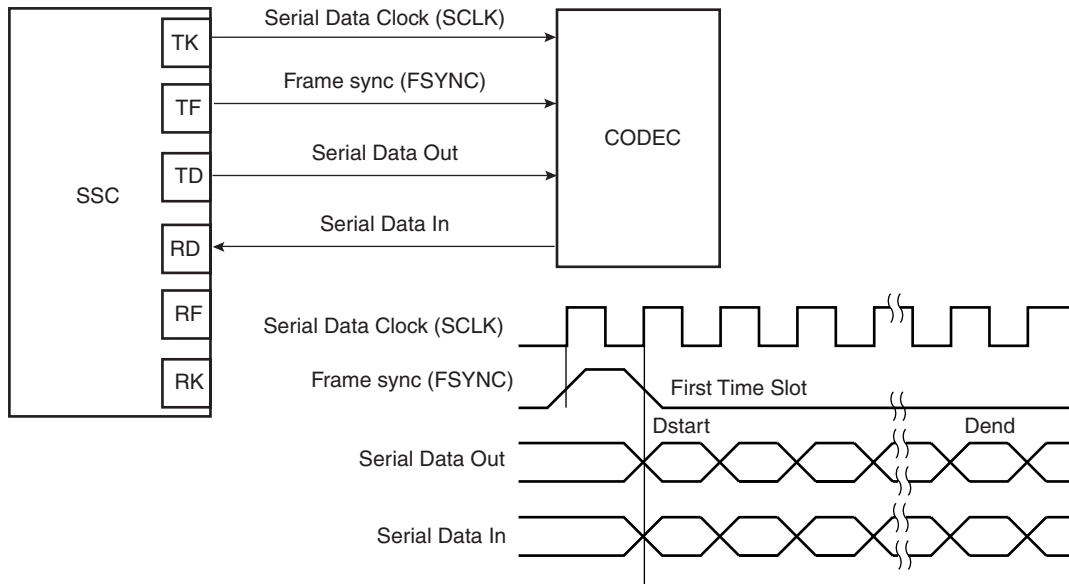
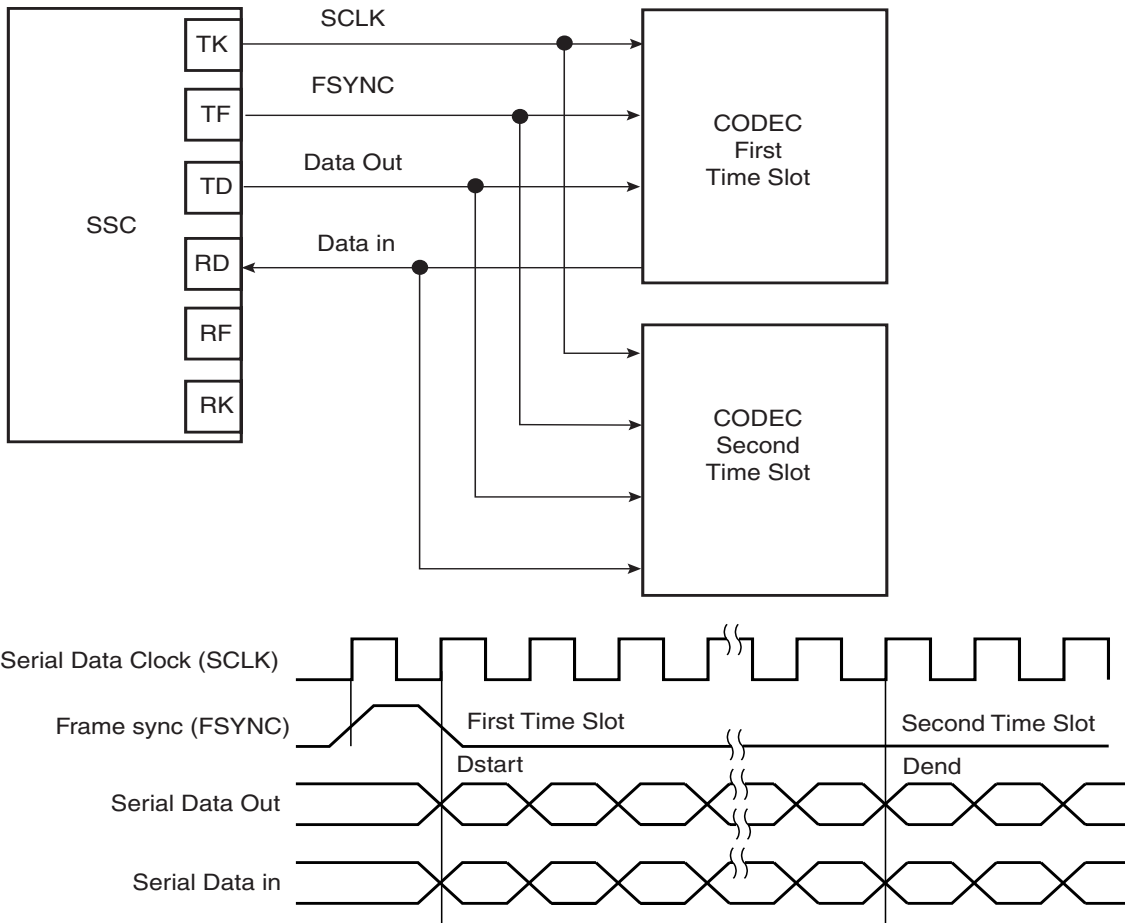


Figure 147. Time Slot Application Block Diagram



## Synchronous Serial Controller (SSC) User Interface

**Table 73.** Synchronous Serial Controller (SSC) 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	Reserved	–	–	–
0x3C	Reserved	–	–	–
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)	–	–	–

## 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 Data Receive if RXDIS is not set<sup>(1)</sup>.

- **RXDIS: Receive Disable**

0: No effect.

1: Disables Data Receive<sup>(1)</sup>.

- **TXEN: Transmit Enable**

0: No effect.

1: Enables Data Transmit if TXDIS is not set<sup>(1)</sup>.

- **TXDIS: Transmit Disable**

0: No effect.

1: Disables Data Transmit<sup>(1)</sup>.

- **SWRST: Software Reset**

0: No effect.

1: Performs a software reset. Has priority on any other bit in SSC\_CR.

Note: 1. Only the data management is affected

## SSC Clock Mode Register

**Name:** SSC\_CMR

**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 x 4095 = MCK/8190.

## 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
STTDLY							
15	14	13	12	11	10	9	8
–	–	–	–	START			
7	6	5	4	3	2	1	0
–	–	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
0x2-0x7	Reserved	

### • CKI: Receive Clock Inversion

0: The data and the Frame Sync signal are sampled on Receive Clock falling edge.

1: The data and the Frame Sync signal are shifted out on Receive Clock rising edge.

CKI does not affect the RK output clock signal.

- **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 input
0x3	Detection of a high level on RF input
0x4	Detection of a falling edge on RF input
0x5	Detection of a rising edge on RF input
0x6	Detection of any level change on RF input
0x7	Detection of any edge on RF input
0x8-0xF	Reserved

- **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.

**Please 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  $2 \times (\text{PERIOD} + 1)$  Receive Clock.



## 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

0x0 is not supported. The value of DATLEN can be set between 0x1 and 0x1F.

The bit stream contains DATLEN + 1 data bits. Moreover, it defines the transfer size performed by the PDC assigned to the Receiver.

If DATLEN is less than or equal to 7, data transfers are in bytes. If DATLEN is between 8 and 15 (included), half-words are transferred. 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. If 0, only 1 data word is transferred. Up to 16 data words can be transferred.

### • 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. Only when FSOS is set on negative or positive pulse.

### • 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 sets RXSYN in the SSC Status Register.

<b>FSEDGE</b>	<b>Frame Sync Edge Detection</b>
0x0	Positive Edge Detection
0x1	Negative Edge Detection

## 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
–	–	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
0x2-0x7	Reserved	

### • CKI: Transmit Clock Inversion

0: The data and the Frame Sync signal are shifted out on Transmit Clock falling edge.

1: The data and the Frame Sync signal are shifted out on Transmit Clock rising edge.

CKI affects only the Transmit Clock and not the output clock signal.

- **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.

**Please 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.



## 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**

0x0 is not supported. The value of DATLEN can be set between 0x1 and 0x1F.

The bit stream contains DATLEN + 1 data bits. Moreover, it defines the transfer size performed by the PDC assigned to the Receiver.

If DATLEN is less than or equal to 7, data transfers are in bytes. If DATLEN is between 8 and 15 (included), half-words are transferred. 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. If 0, only 1 data word is transferred and up to 16 data words can be transferred.

- **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. If 0, the Transmit Frame Sync signal is generated during one Transmit Clock period and up to 16 clock period pulse length is possible.

- **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.

- **FSEDGE: Frame Sync Edge Detection**

Determines which edge on frame sync sets TXSYN (Status Register).

<b>FSEDGE</b>	<b>Frame Sync Edge Detection</b>
0x0	Positive Edge Detection
0x1	Negative Edge Detection



## SSC Receive Holding Register

**Name:** SSC\_RHR

**Access Type:** Read-only

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

- **RDAT: Receive Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC\_RFMR.

## SSC Transmit Holding Register

**Name:** SSC\_THR

**Access Type:** Write only

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

- **TDAT: Transmit Data**

Right aligned regardless of the number of data bits defined by DATLEN in SSC\_TFMR.

## SSC Receive Synchronization Holding Register

Name: SSC\_RSHR

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
RSDAT							
7	6	5	4	3	2	1	0
RSDAT							

- **RSDAT: Receive Synchronization Data**

Right aligned regardless of the number of data bits defined by FSLEN in SSC\_RFMR.

## 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**

Right aligned regardless of the number of data bits defined by FSLEN in SSC\_TFMR.

## SSC Status Register

**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	–	–
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.

1: SSC\_THR is empty.

- **TXEMPTY: Transmit Empty**

0: Data remains in SSC\_THR or is currently transmitted from Transmit Shift Register.

1: Last data written in SSC\_THR has been loaded in Transmit Shift Register and transmitted by it.

- **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.

- **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: A Rx Sync has not occurred since the last read of the Status Register.

1: A Rx Sync has occurred since the last read of the Status Register.



- **TXEN: Transmit Enable**

0: Transmit data is disabled.

1: Transmit data is enabled.

- **RXEN: Receive Enable**

0: Receive data is disabled.

1: Receive data is enabled.

## SSC Interrupt Enable Register

**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	–	–
7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready**
- **TXEMPTY: Transmit Empty**
- **ENDTX: End of Transmission**
- **TXBUFE: Transmit Buffer Empty**
- **RXRDY: Receive Ready**
- **OVRUN: Receive Overrun**
- **ENDRX: End of Reception**
- **RXBUFF: Receive Buffer Full**
- **TXSYN: Tx Sync**
- **RXSYN: Rx Sync**

0: No effect.

1: Enables the corresponding interrupt.

## SSC Interrupt Disable Register

**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	–	–
7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready**
- **TXEMPTY: Transmit Empty**
- **ENDTX: End of Transmission**
- **TXBUFE: Transmit Buffer Empty**
- **RXRDY: Receive Ready**
- **OVRUN: Receive Overrun**
- **ENDRX: End of Reception**
- **RXBUFF: Receive Buffer Full**
- **TXSYN: Tx Sync**
- **RXSYN: Rx Sync**

0: No effect.

1: Disables the corresponding interrupt.



## SSC Interrupt Mask Register

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	–	–
7	6	5	4	3	2	1	0
RXBUFF	ENDRX	OVRUN	RXRDY	TXBUFE	ENDTX	TXEMPTY	TXRDY

- **TXRDY: Transmit Ready**
- **TXEMPTY: Transmit Empty**
- **ENDTX: End of Transmission**
- **TXBUFE: Transmit Buffer Empty**
- **RXRDY: Receive Ready**
- **OVRUN: Receive Overrun**
- **ENDRX: End of Reception**
- **RXBUFF: Receive Buffer Full**
- **TXSYN: Tx Sync**
- **RXSYN: Rx Sync**

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

## Timer/Counter (TC)

### 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.

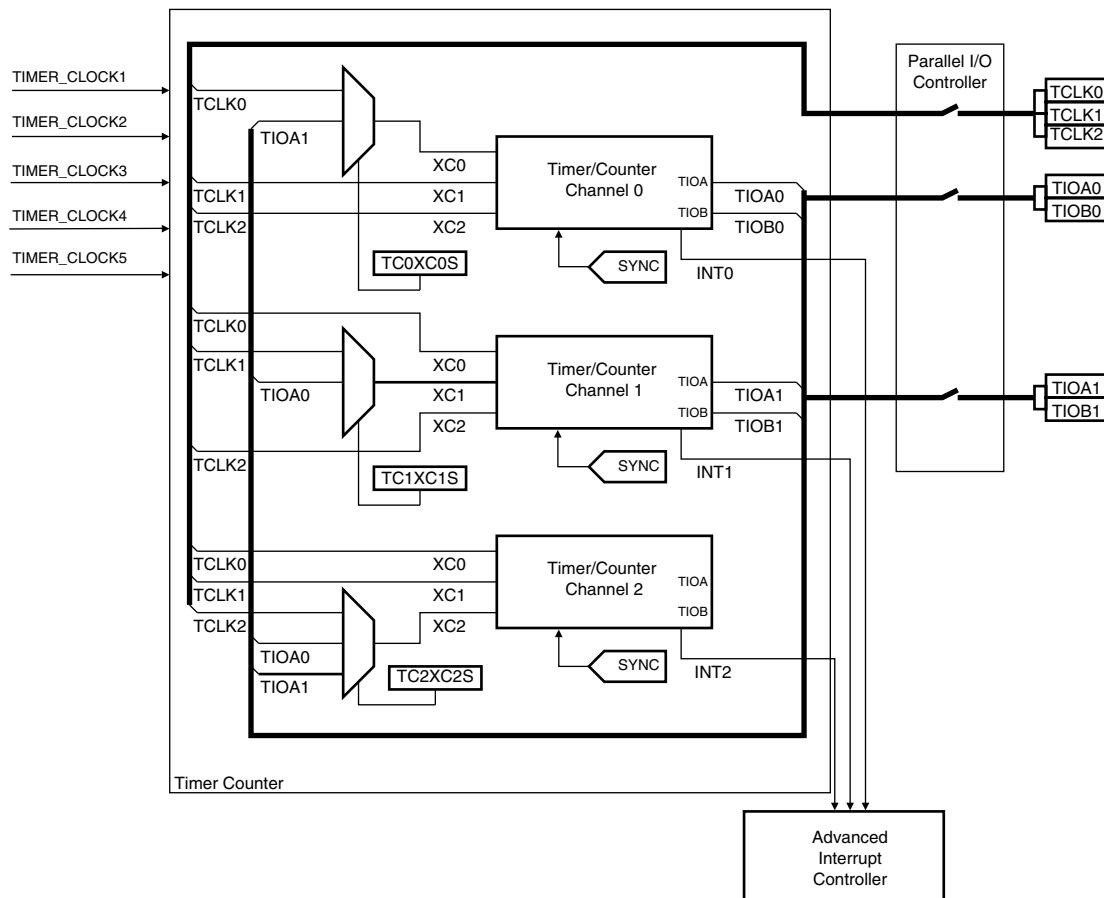
The Timer/Counter block has two global registers which act upon all three TC channels.

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.

### Block Diagram

Figure 148. Timer/Counter Block Diagram



**Table 74.** 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

## Pin Name List

**Table 75.** TC pin list

Pin Name	Description	Type
TCLK0-TCLK2	External Clock Input	Input
TIOA0-TIOA1	I/O Line A	I/O
TIOB0-TIOB1	I/O Line B	I/O

## Product Dependencies

### 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.

### 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.

### 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.

## Functional Description

### TC Description

The three channels of the Timer/Counter are independent and identical in operation. The registers for channel programming are listed in Table 77 on page 354.

### 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.

## 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 I/O signals TIOA0 or TIOA1 for chaining by programming the TC\_BMR (Block Mode). See Figure 149.

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

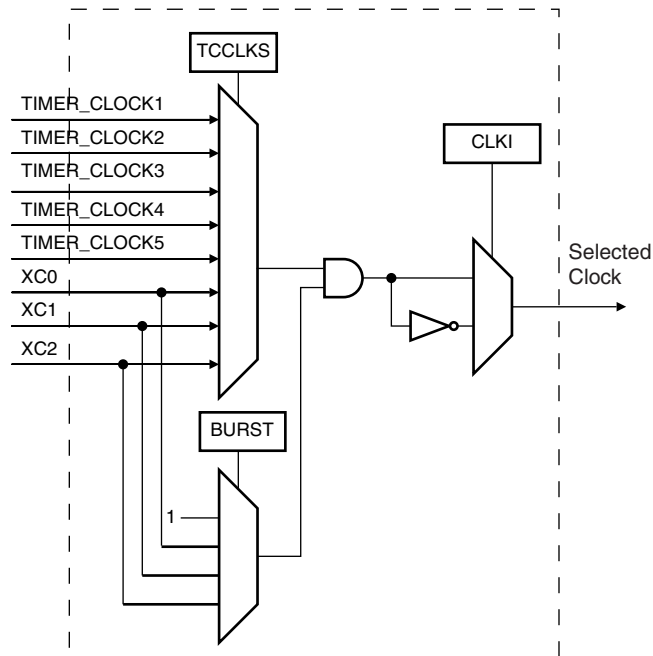
This selection is made by the TCCLKS bits in the TC Channel Mode Register.

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).

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 149.** Clock Selection



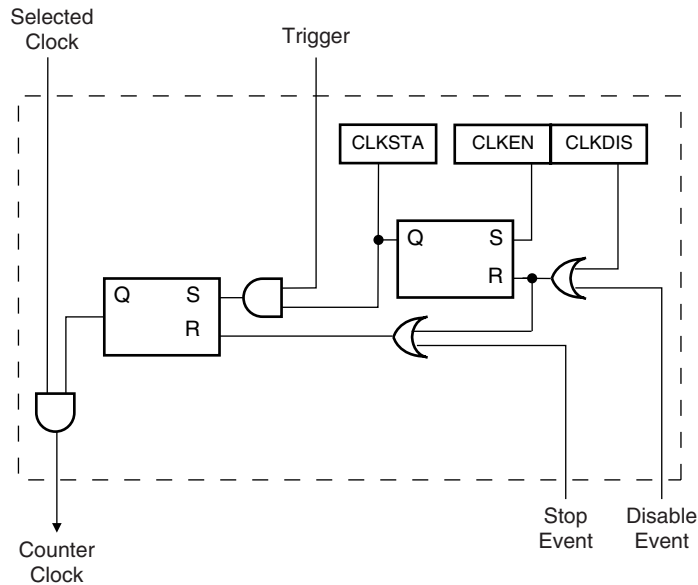
## Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 150.

- 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 150.** Clock Control



### 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.

### 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 CPCTRIG 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 ENETRIG 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.

## 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 151 shows the configuration of the TC channel when programmed in Capture Mode.

## 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.

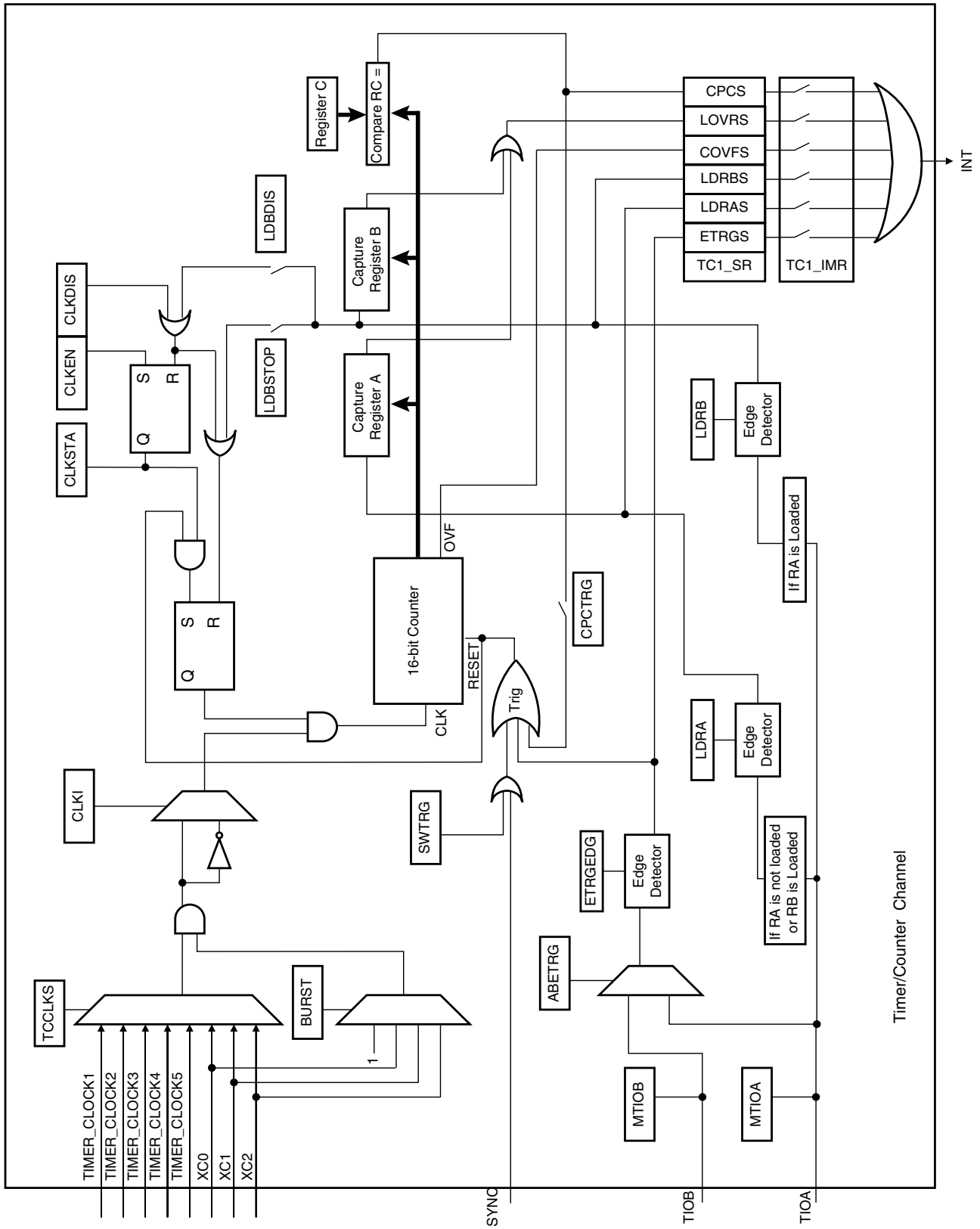
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS) in TC\_SR (Status Register). In this case, the old value is overwritten.

## Trigger Conditions

In addition to the SYNC signal, the software trigger and the RC compare trigger, an external trigger can be defined.

The ABETRG bit in 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 151. Capture Mode



## 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 152 shows the configuration of the TC channel when programmed in Waveform Operating Mode.

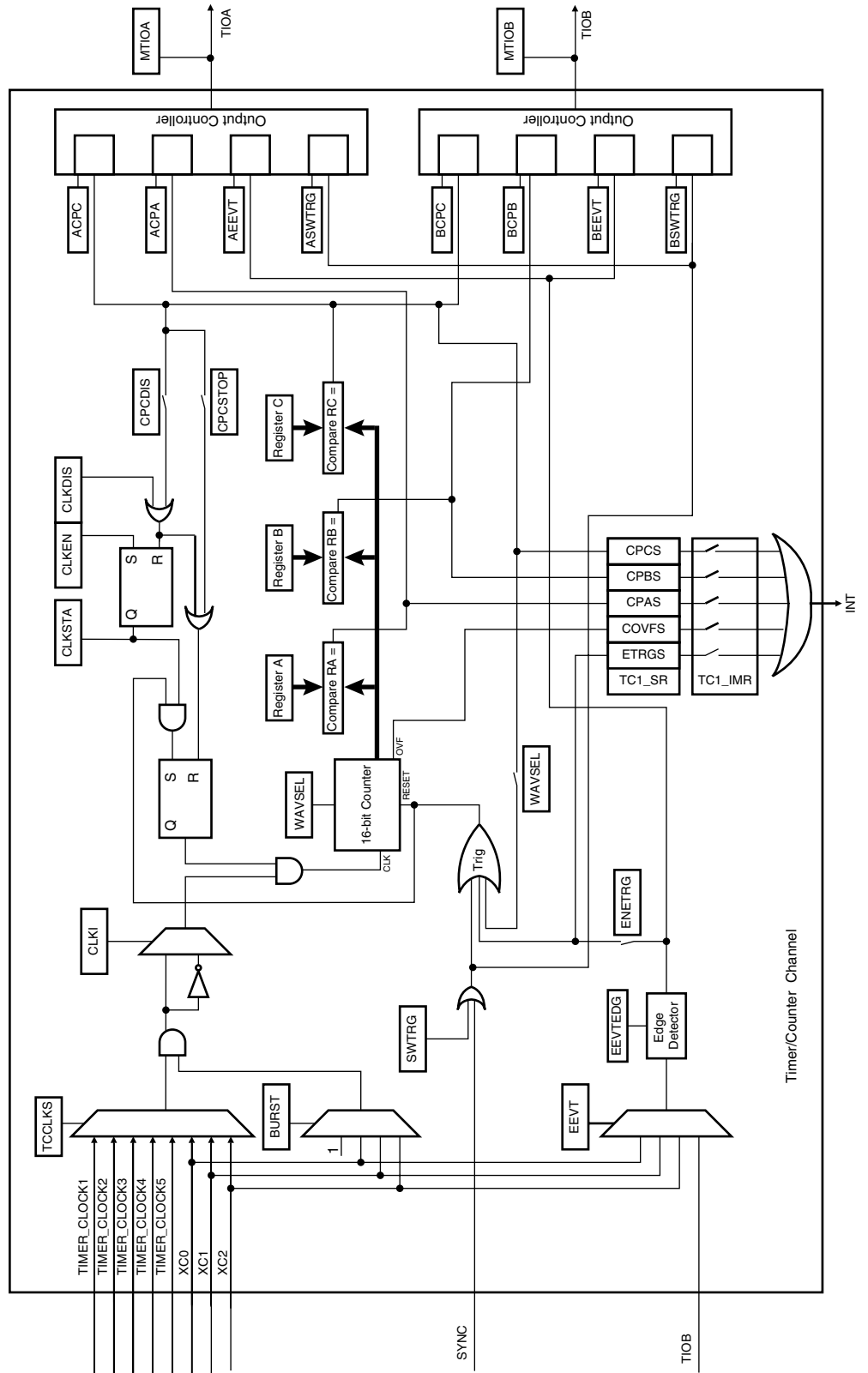
## Waveform Selection

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 152. Waveform Mode



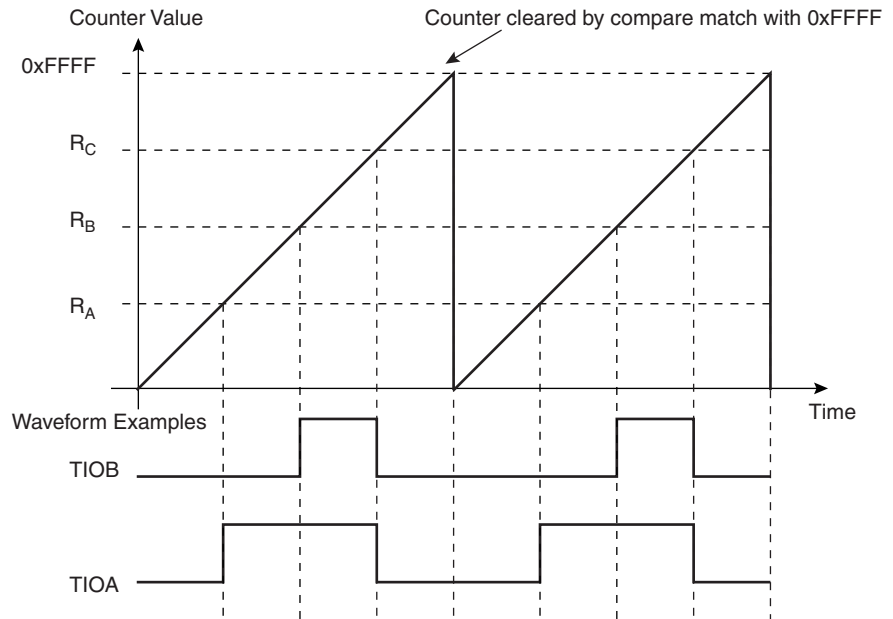
WAVSEL = 00

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 153.

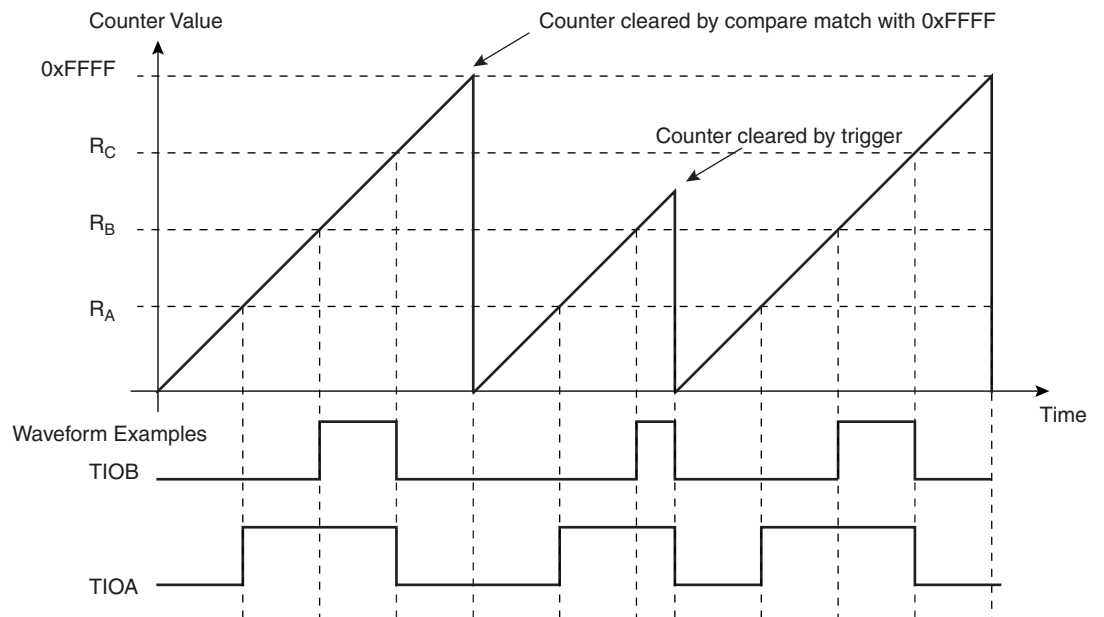
An external event trigger or a software trigger can reset the value of TC\_CV. It is important to note that the trigger may occur at any time. See Figure 154.

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 153.** WAVSEL= 00 without trigger



**Figure 154.** WAVSEL= 00 with trigger



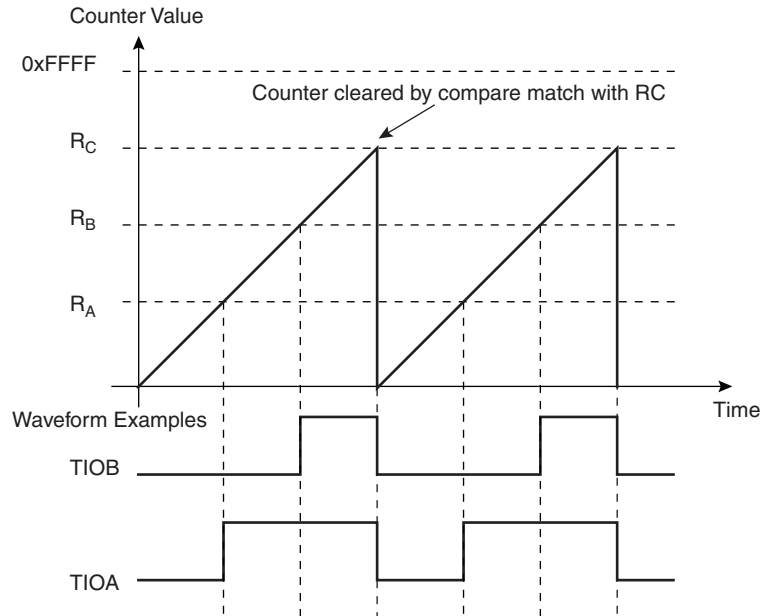
WAVSEL = 10

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 155.

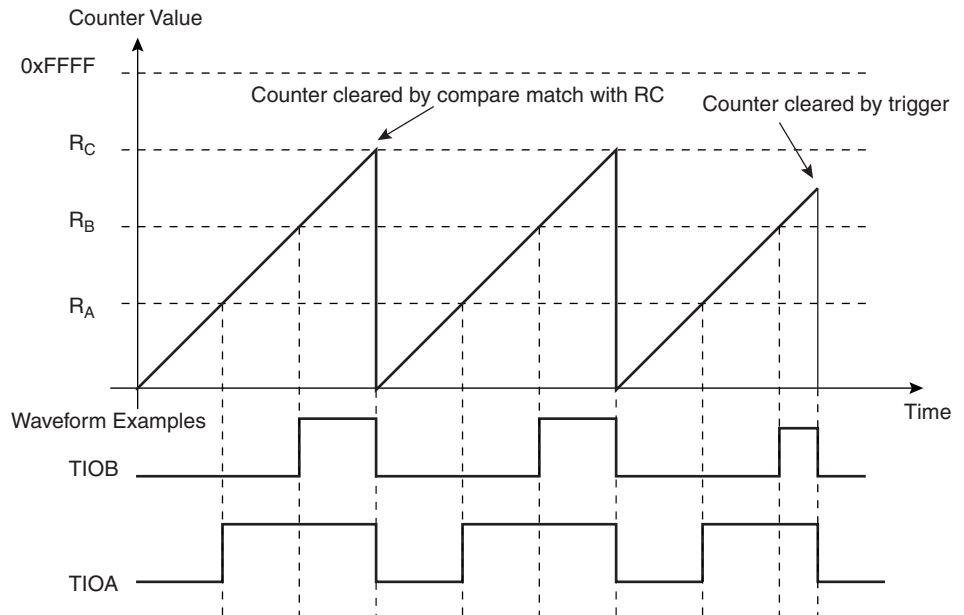
It is important to note that TC\_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See Figure 156.

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 155.** WAVSEL = 10 Without Trigger



**Figure 156.** WAVSEL = 10 With Trigger



WAVSEL = 01

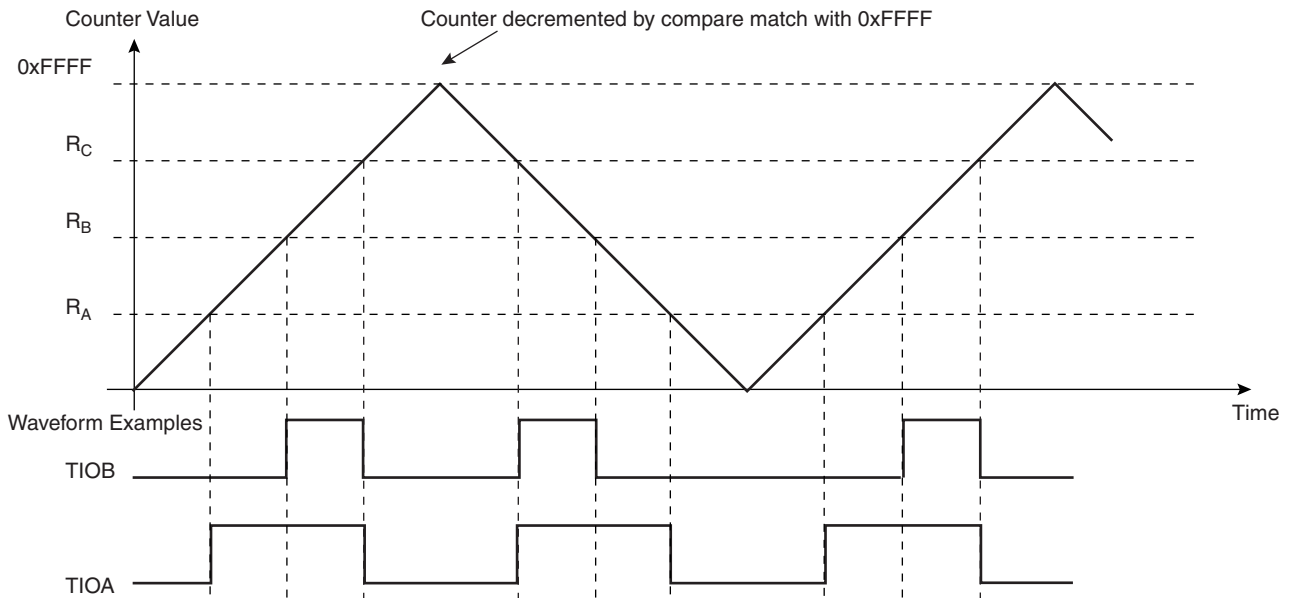
When WAVSEL = 01, the value of TC\_CV is incremented from 0 to 0xFFFF. Once 0xFFFF is reached, the value of TC\_CV is decremented to 0, then re-incremented to 0xFFFF and so on. See Figure 157.

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 158.

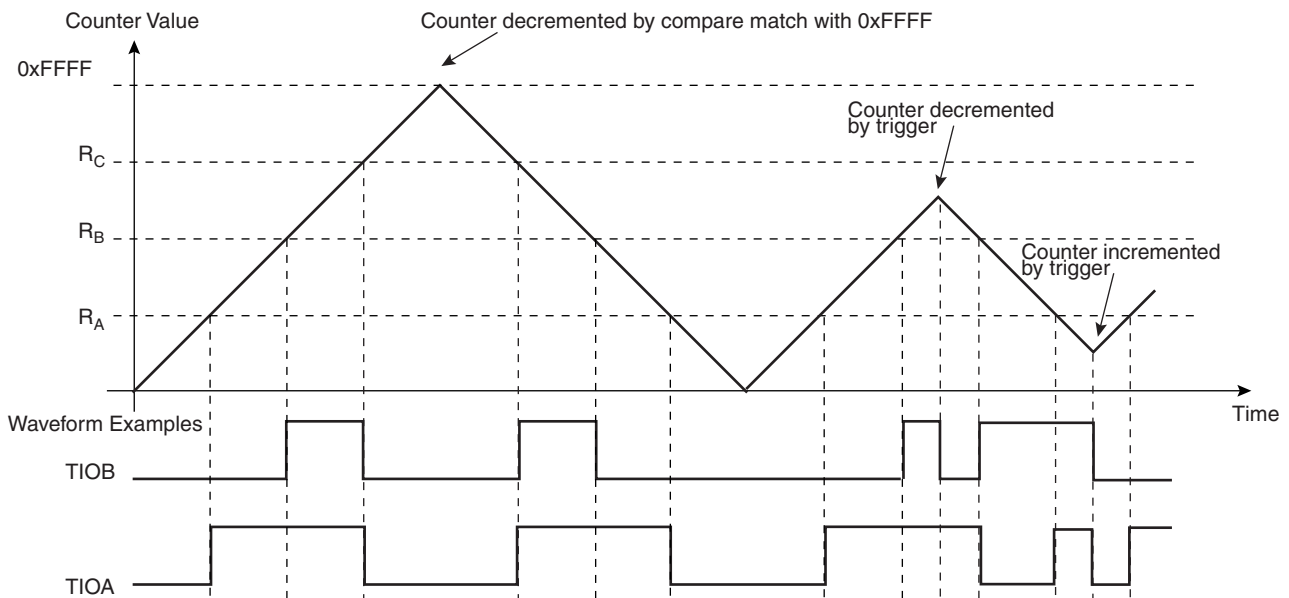
RC Compare cannot be programmed to generate a trigger in this configuration.

At the same time, RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 157. WAVSEL = 01 Without Trigger**



**Figure 158. WAVSEL = 01 With Trigger**



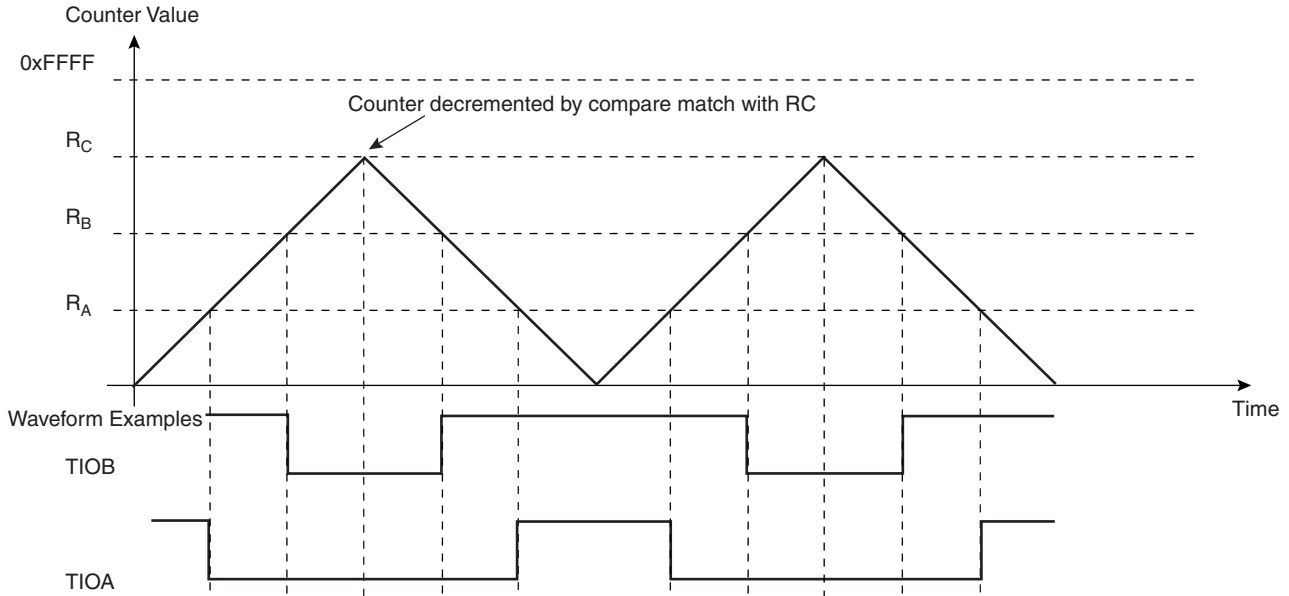
WAVSEL = 11

When WAVSEL = 11, the value of TC\_CV is incremented from 0 to RC. Once RC is reached, the value of TC\_CV is decremented to 0, then re-incremented to RC and so on. See Figure 159.

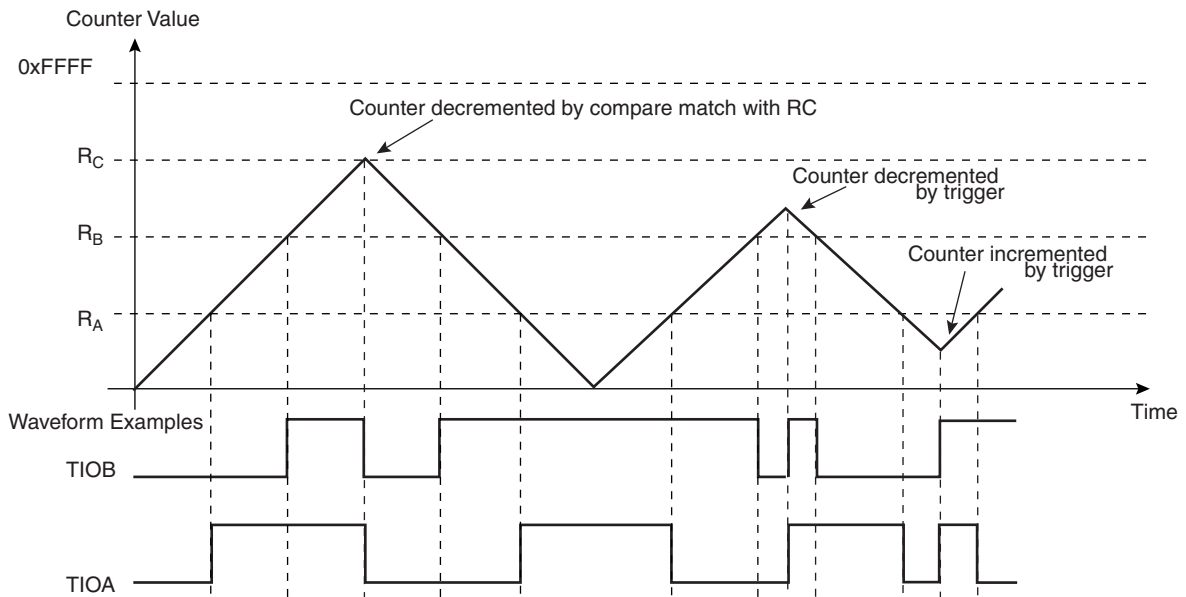
A trigger such as an external event or a software trigger can modify TC\_CV at any time. If a trigger occurs while TC\_CV is incrementing, TC\_CV then decrements. If a trigger is received while TC\_CV is decrementing, TC\_CV then increments. See Figure 160.

RC Compare can stop the counter clock (CPCSTOP = 1) and/or disable the counter clock (CPCDIS = 1).

**Figure 159. WAVSEL = 11 Without Trigger**



**Figure 160. WAVSEL = 11 With Trigger**





## 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 parameter EEVT parameter in TC\_CMR selects the external trigger. The EEVTEDG parameter defines the trigger edge for each of the possible external triggers (rising, falling or both). If EEVTEDG is cleared (none), no external event is defined.

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as an output and the 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 ENETRIG 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.

## 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.

## Timer/Counter (TC) User Interface

### Global Register Mapping

**Table 76.** Timer/Counter (TC) Global Register Map

Offset	Channel/Register	Name	Access	Reset Value
0x00	TC Channel 0		See Table 77	
0x40	TC Channel 1		See Table 77	
0x80	TC Channel 2		See Table 77	
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 77. The offset of each of the channel registers in Table 77 is in relation to the offset of the corresponding channel as mentioned in Table 77.

### Channel Memory Mapping

**Table 77.** Timer/Counter (TC) Channel Memory Mapping

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
0x30-0xFC	Reserved	–	–	–

Note: 1. Read only if WAVE = 0

## 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.

## 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

- **TC1XC1S: External Clock Signal 1 Selection**

TC1XC1S		Signal Connected to XC1
0	0	TCLK1
0	1	none
1	0	TIOA0

- **TC2XC2S: External Clock Signal 2 Selection**

TC2XC2S		Signal Connected to XC2
0	0	TCLK2
0	1	none
1	0	TIOA0
1	1	TIOA1

### 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.

## 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

## 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.

- **EEVTEG: External Event Edge Selection**

EEVTEG		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.

- **ENETR: 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



- **AAEVT: External Event Effect on TIOA**

AAEVT		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

## 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.

## 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.

## 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.

## 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.

## 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.



## 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.

## 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.

## 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.



## Pulse Width Modulation Controller (PWM)

### Overview

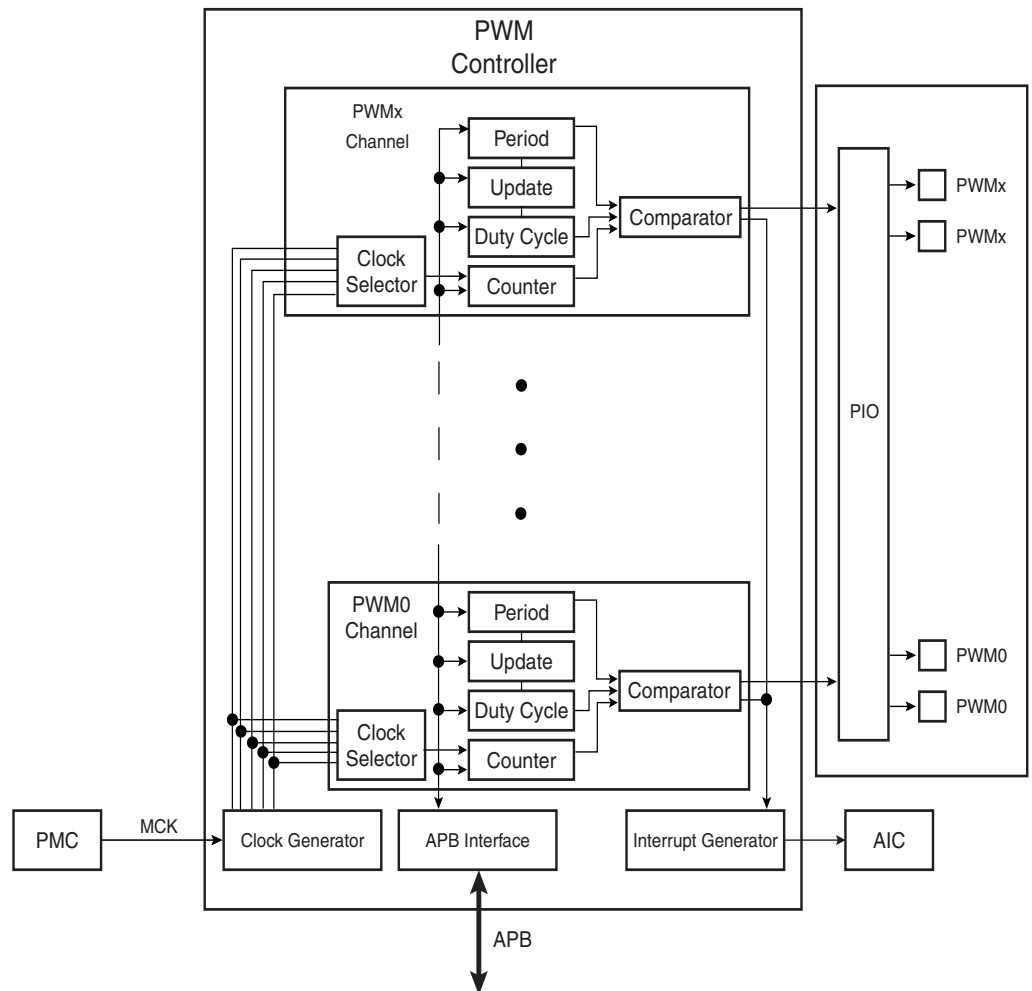
The PWM macrocell controls several channels independently. Each channel controls one square output waveform. Characteristics of the output waveform such as period, duty-cycle and polarity are configurable through the user interface. Each channel selects and uses one of the clocks provided by the clock 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.

### Block Diagram

Figure 161. Pulse Width Modulation Controller Block Diagram



## I/O Lines Description

Each channel outputs one waveform on one external I/O line.

**Table 78.** I/O Line Description

Name	Description	Type
PWMx	PWM Waveform Output for channel x	Output

## Product Dependencies

### 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.

### 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.

### 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.

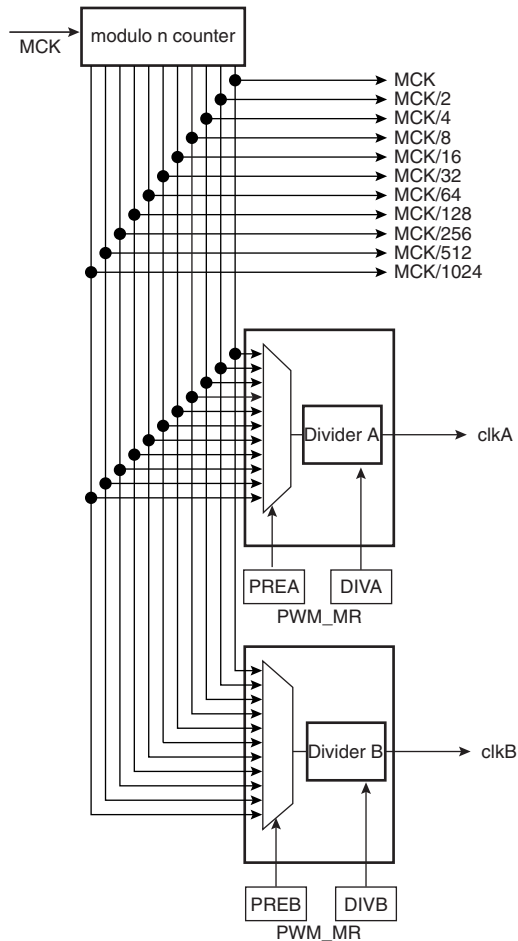
## Functional Description

The PWM macrocell is primarily composed of a clock generator module and 4 channels.

- Clocked by the system clock, MCK, the clock generator module provides 13 clocks.
- Each channel can independently choose one of the clock generator outputs.
- Each channel generates an output waveform with attributes that can be defined independently for each channel through the user interface registers.

## PWM Clock Generator

**Figure 162.** 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).

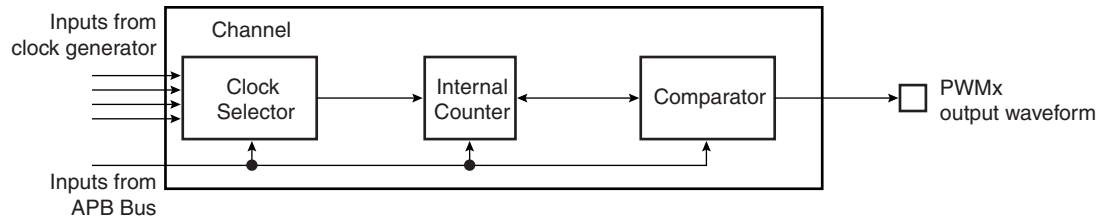
After a reset of the PWM controller, DIVA (DIVB) and PREA (PREB) in the PWM Mode register are set to 0. This implies that after reset clkA (clkB) are turned off.

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.

## PWM Channel

### Block Diagram

**Figure 163.** Functional View of the Channel Block Diagram



Each of the 4 channels is composed of three blocks:

- A clock selector which selects one of the clocks provided by the clock generator described in Section “PWM Clock Generator” on page 371.
- An internal counter clocked by the output of the clock selector. This internal counter is incremented or decremented according to the channel configuration and comparators events. The size of the internal counter is 16 bits.
- A comparator used to generate events according to the internal counter value. It also computes the PWMx output waveform according to the configuration.

### 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:

$$\frac{(CPRD \times DIVA)}{MCK} \text{ or } \frac{(CPRD \times DIVB)}{MCK}$$

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:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

- 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:

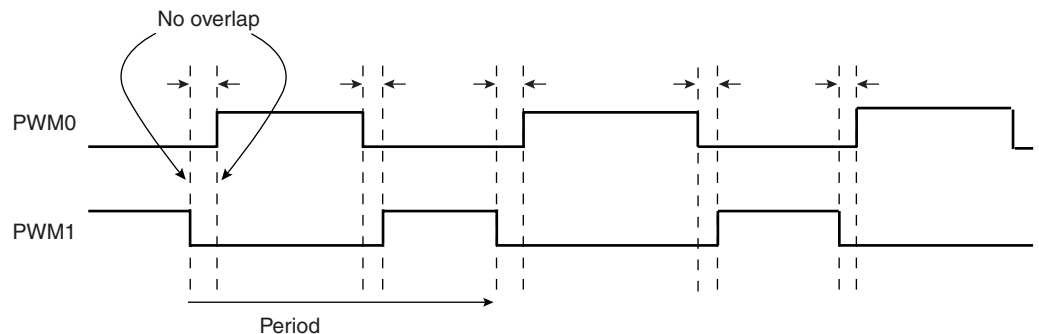
$$\text{duty cycle} = \frac{(\text{period} - 1 / \text{fchannel\_x\_clock} \times CDTY)}{\text{period}}$$

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 164.** Non Overlapped Center Aligned Waveforms



Note: 1. See Figure 165 on page 375 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

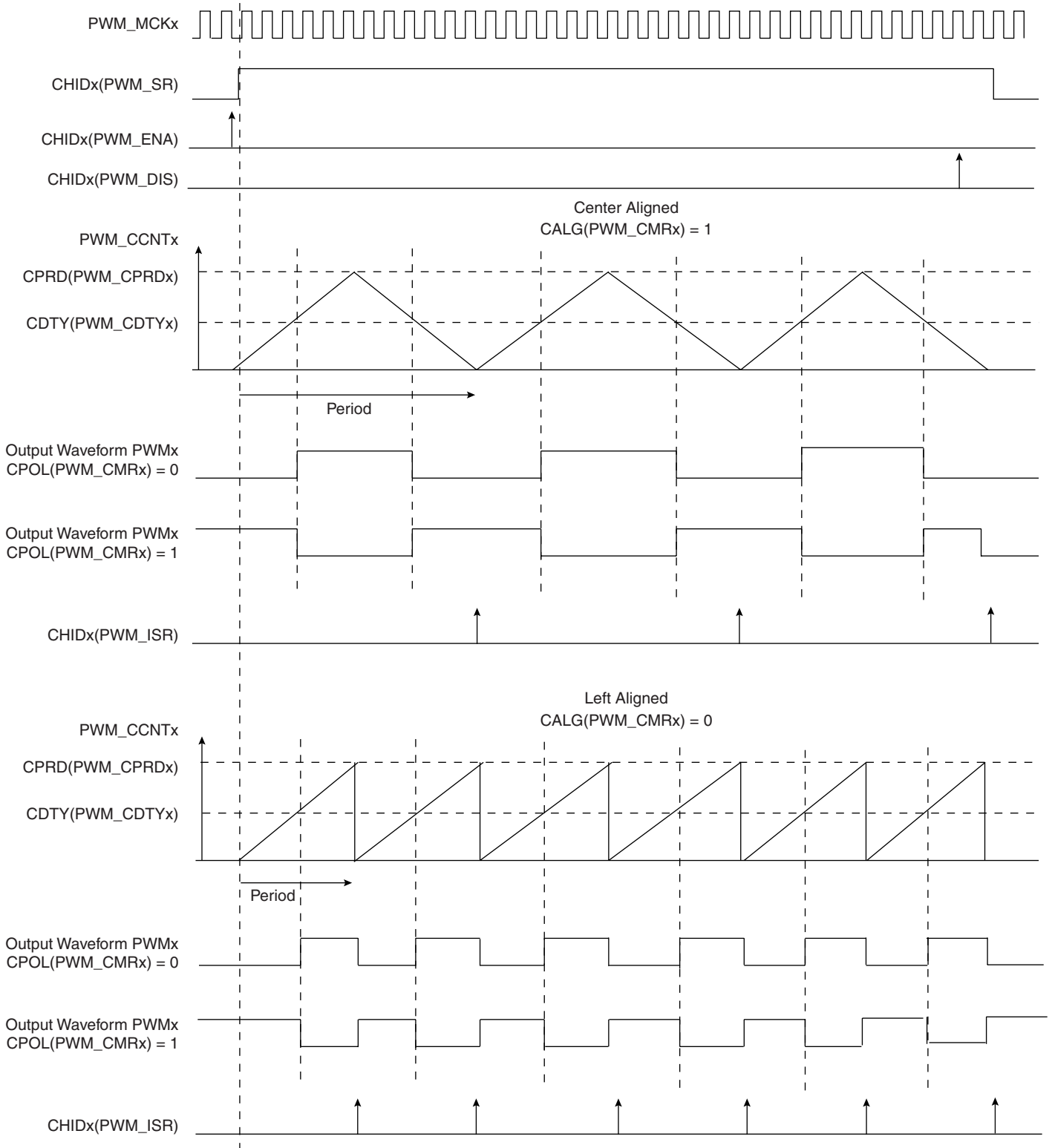


Waveforms are fixed at 1 (once the channel is enabled) when:

- 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 165. Waveform Properties**



## PWM Controller Operations

### 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.

### 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.

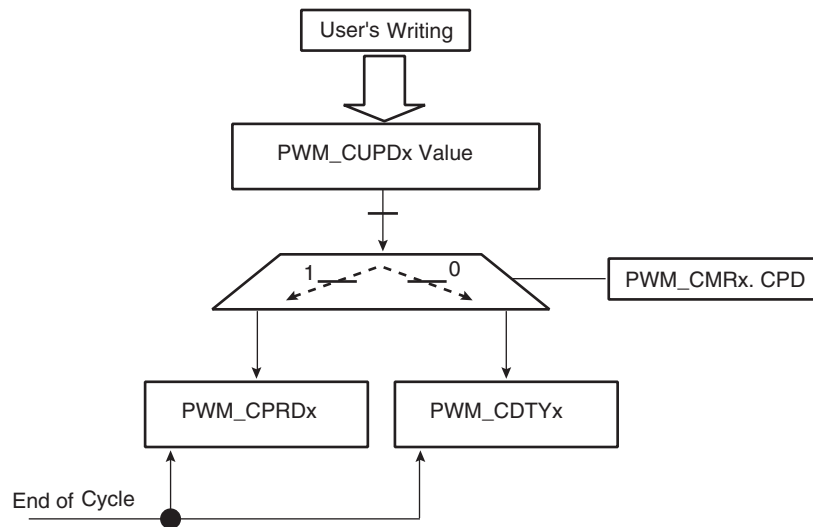
### Changing the Duty Cycle or the Period

It is possible to modulate the output waveform duty cycle or period.

To prevent an unexpected output waveform when modifying the waveform parameters while the channel is still enabled, PWM\_CPRDx and PWM\_CDTYx registers are double buffered. 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. According to the CPD field in the PWM\_CMRx register, PWM\_CUPDx either updates the PWM\_CPRDx or PWM\_CDTYx.



**Figure 166.** Synchronized Period or Duty Cycle Update



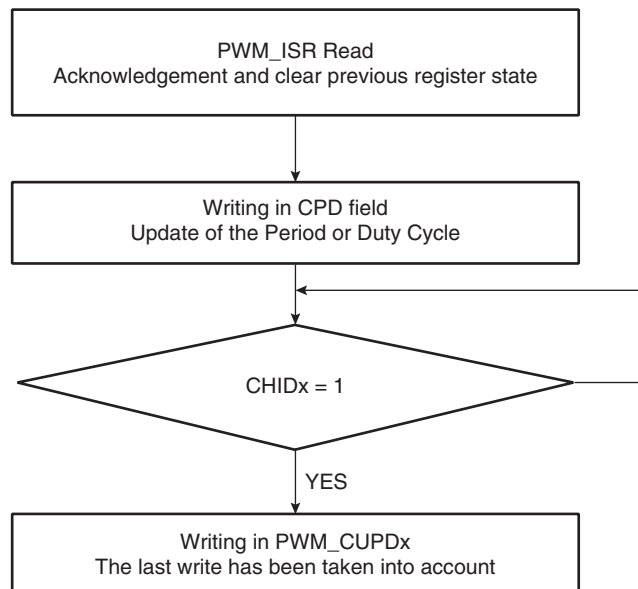
To prevent overwriting the PWM\_CUPDx by software, the user can use status events in order to synchronize his software. Two methods are possible. In both, the user must enable the dedicated interrupt in PWM\_IER at PWM Controller level.

The first method (polling method) consists of reading the relevant status bit in PWM\_ISR Register according to the enabled channel(s). See Figure 167.

The second method uses an Interrupt Service Routine associated with the PWM channel.

Note: Reading the PWM\_ISR register automatically clears CHIDx flags.

**Figure 167.** Polling Method



## PWM User Interface

### PWM Register Mapping

**Table 79.** 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 - 0xF8	Reserved	-	-	-
0xFC	Version Register	PWM_VERSION	Read-only	0x- <sup>(1)</sup>
0x100 - 0x1FC	Reserved			
0x200	Channel 0 Mode Register	PWM_CMRO	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_CMRI	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	-
...	...	...	...	...

Note: 1. Values in the Version Register vary with the version of the IP block implementation.

## 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

## 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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = No effect.

1 = Enable PWM output for channel x.

## 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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = No effect.

1 = Disable PWM output for channel x.

## 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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID**

0 = PWM output for channel x is disabled.

1 = PWM output for channel x is enabled.

## 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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID.**

0 = No effect.

1 = Enable interrupt for PWM channel x.

## 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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID.**

0 = No effect.

1 = Disable interrupt for PWM channel x.

## 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
–	–	–	–	CHID3	CHID2	CHID1	CHID0

- **CHIDx: Channel ID.**

0 = Interrupt for PWM channel x is disabled.

1 = Interrupt for PWM channel x is enabled.

## 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
–	–	–	–	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.



## 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.



## PWM Channel Duty Cycle Register

**Register Name:** PWM\_CDTYx

**Access Type:** Read/Write

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

Only the first 16 bits (internal channel counter size) are significant.

- **CDTY: Channel Duty Cycle**

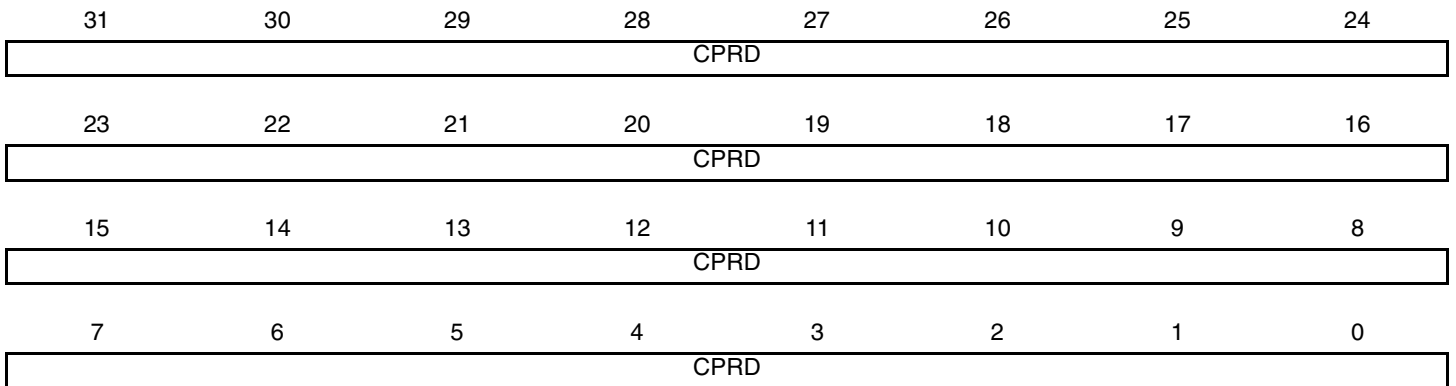
Defines the waveform duty cycle. This value must be defined between 0 and CPRD (PWM\_CPRx).



## PWM Channel Period Register

Register Name: PWM\_CPRDx

Access Type: Read/Write



Only the first 16 bits (internal channel counter size) are significative.

### • 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:

$$\frac{(2 \times CPRD \times DIVA)}{MCK} \text{ or } \frac{(2 \times CPRD \times DIVB)}{MCK}$$

## 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.

## 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 16 bits (internal channel counter size) are significative.

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.



## PWM Version Register

**Register Name:** PWM\_VERSION

**Access Type:** Read-only

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

- **VERSION**

Reserved. Value subject to change. No functionality associated. This is the Atmel internal version of the macrocell.

- **MFN**

Reserved. Value subject to change. No functionality associated.

## Analog-to-digital Converter (ADC)

### 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 up to eight analog lines. The conversions extend from 0V to ADVREF.

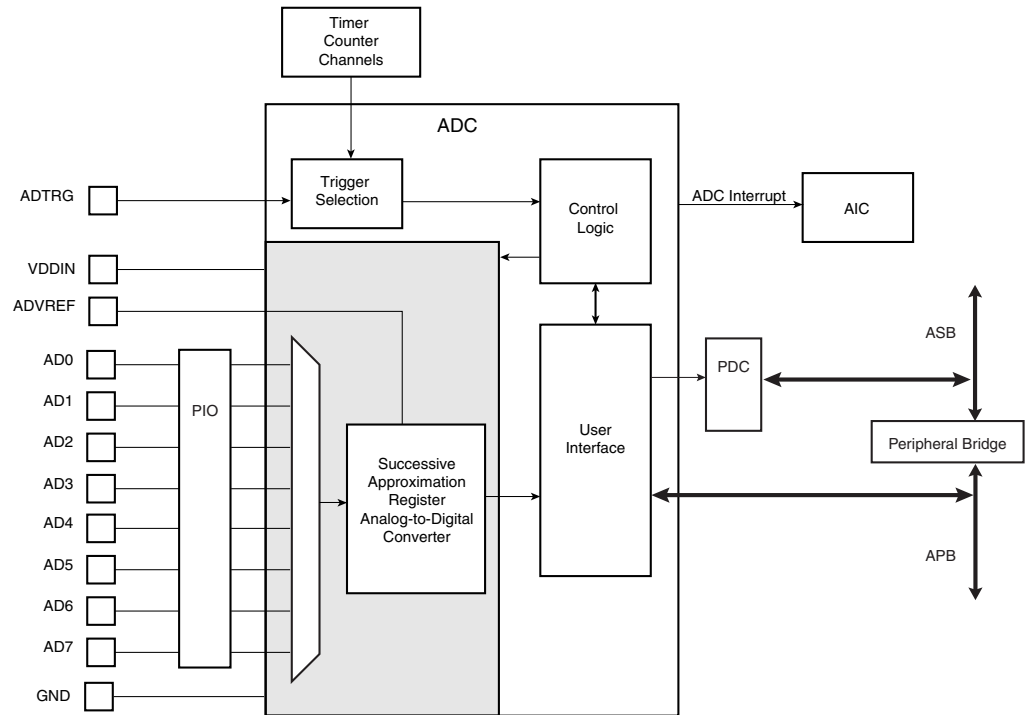
The ADC supports an 8-bit or 10-bit resolution mode, and conversion results are reported in a common register for all channels, as well as in a channel-dedicated register. Software trigger, external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

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.

### Block Diagram

Figure 168. Analog-to-Digital Converter Block Diagram



## Signal Description

**Table 80.** ADC Pin Description

Pin Name	Description
VDDIN	Analog power supply
ADVREF	Reference voltage
AD0 - AD7	Analog input channels
ADTRG	External trigger

## Product Dependencies

### 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.

### 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.

### Analog Inputs

The pins AD0 to AD7 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.

### 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.

### 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.

### Conversion Performances

For performance and electrical characteristics of the ADC, see the DC Characteristics section.

## Functional Description

### 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 397 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.

### 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.

### 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.

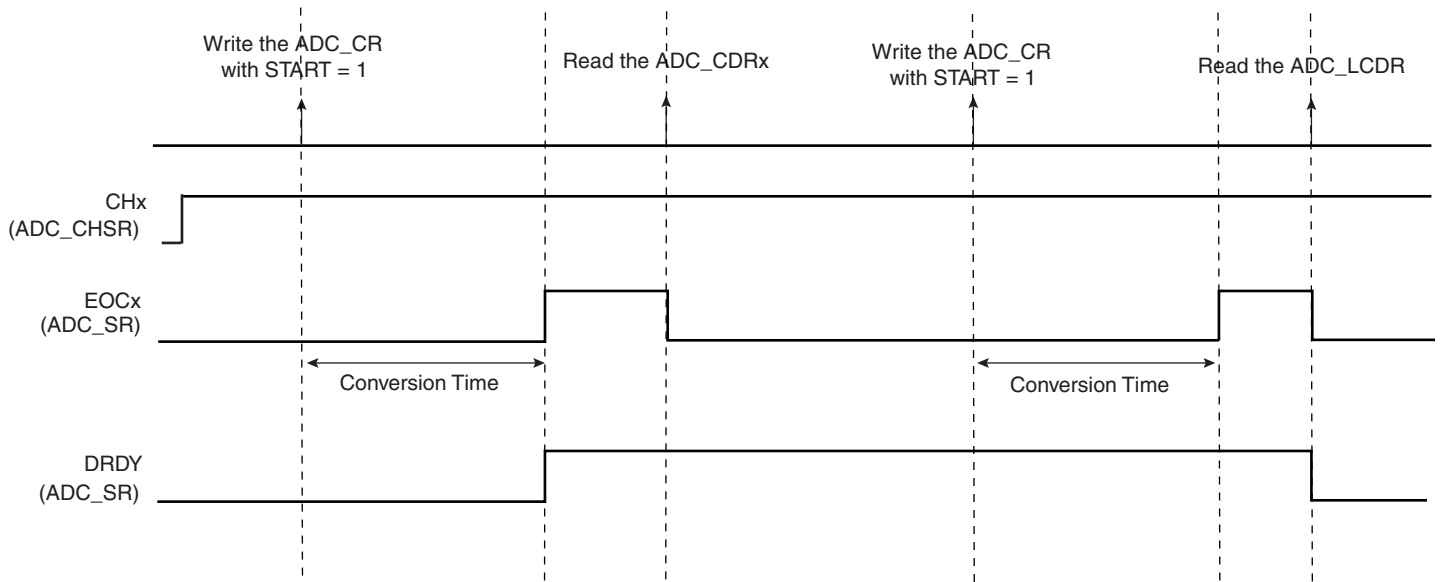
### Conversion Results

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\_LCDR).

The channel EOC bit in the Status Register (ADC\_SR) is set and the DRDY is set. In the case of a connected PDC channel, DRDY rising triggers a data transfer request. In any case, either EOC and DRDY can trigger an interrupt.

Reading one of the ADC\_CDR registers clears the corresponding EOC bit. Reading ADC\_LCDR clears the DRDY bit and the EOC bit corresponding to the last converted channel.

**Figure 169. EOCx and DRDY Flag Behavior**



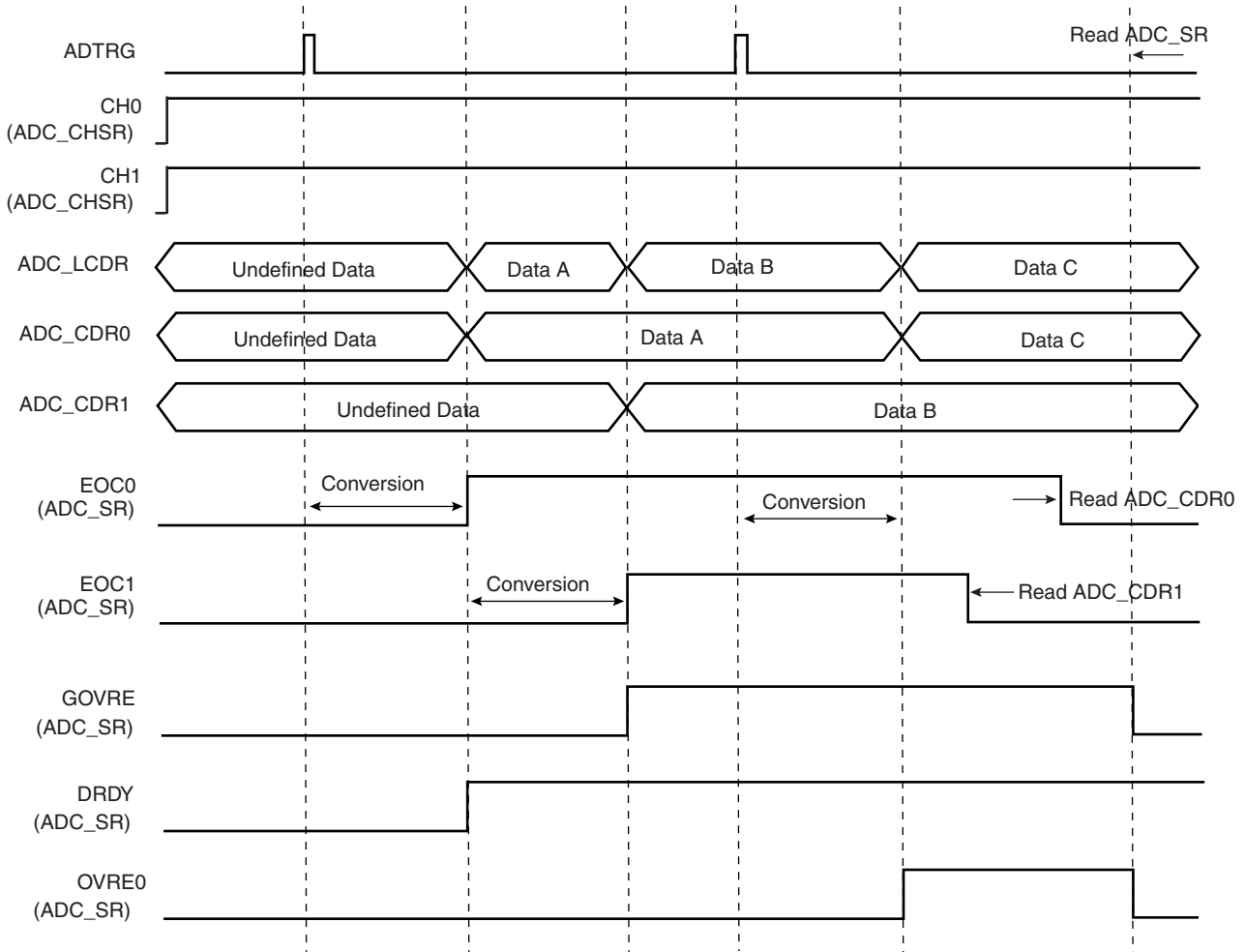


If the ADC\_CDR is not read before further incoming data is converted, the corresponding Overrun Error (OVRE) flag is set in the Status Register (ADC\_SR).

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 170. 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.

## 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.

## 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.

## 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 DC Characteristics in the product datasheet.

## Analog-to-digital Converter (ADC) User Interface

**Table 81.** 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
0x38	Channel Data Register 2	ADC_CDR2	Read-only	0x00000000
0x3C	Channel Data Register 3	ADC_CDR3	Read-only	0x00000000
0x40	Channel Data Register 4	ADC_CDR4	Read-only	0x00000000
0x44	Channel Data Register 5	ADC_CDR5	Read-only	0x00000000
0x48	Channel Data Register 6	ADC_CDR6	Read-only	0x00000000
0x4C	Channel Data Register 7	ADC_CDR7	Read-only	0x00000000
0x50 - 0xFC	Reserved	–	–	–

## 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.

## 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 Output of the Timer Counter Channel 0
0	0	1	TIOA Output of the Timer Counter Channel 1
0	1	0	Reserved
0	1	1	Reserved
1	0	0	Reserved
1	0	1	Reserved
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**

$$\text{Sample \& Hold Time} = (\text{SHTIM} + 1) / \text{ADCClock}$$

## 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.

## 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.

## 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.



## 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 Overrun Error occurred since the last read of ADC\_SR.

1 = At least one 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.

## 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.

## 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.

## 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.

## AT91SAM7S32 Electrical Characteristics

### Absolute Maximum Ratings

**Table 82.** 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 (VDDCORE, and VDDPLL) .....	1.95V
Maximum Operating Voltage (VDDIO, VDDIN and VDDFLASH).....	3.6V
Total DC Output Current on all I/O lines .....	100 mA

\*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## 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  $T_J = 100^{\circ}\text{C}$ .

**Table 83.** DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units	
$V_{VDDCORE}$	DC Supply Core		1.65		1.95	V	
$V_{VDDPLL}$	DC Supply PLL		1.65		1.95	V	
$V_{VDDIO}$	DC Supply I/Os		3.30		3.6	V	
$V_{VDDFLASH}$	DC Supply Flash		3.0		3.6	V	
$V_{IL}$	Input Low-level Voltage		-0.3		0.8	V	
$V_{IH}$	Input High-level Voltage		2.0		5.5	V	
$V_{OL}$	Output Low-level Voltage	$I_O = 8\text{ mA}$			0.4	V	
$V_{OH}$	Output High-level Voltage	$I_O = 8\text{ mA}$	$V_{DDIO} - 0.4$			V	
$I_{LEAK}$	Input Leakage Current	Pull-up resistors disabled (Typ: $T_A = 25^{\circ}\text{C}$ , Max: $T_A = 85^{\circ}\text{C}$ )		20	200	nA	
$I_{PULLUP}$	Input Pull-up Current		143	321	600	$\mu\text{A}$	
$C_{IN}$	Input Capacitance	48-LQFP Package			13.8	pF	
$I_{SC}$	Static Current	On $V_{VDDCORE} = 1.85\text{V}$ , MCK = 0 Hz	$T_A = 25^{\circ}\text{C}$		26	50	$\mu\text{A}$
		All inputs driven at 1 (including TMS, TDI, TCK, NRST)	$T_A = 85^{\circ}\text{C}$		260	500	
$I_O$	Output Current	PA0-PA3			16	mA	
		PA4-PA20			8	mA	

**Table 84.** 1.8V Voltage Regulator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{VDDIN}$	Supply Voltage		3.0	3.3	3.6	V
$V_{VDDOUT}$	Output Voltage		1.81	1.85	1.89	V
$I_{VDDIN}$	Current consumption	After startup, no load		90		$\mu\text{A}$
		During startup, no load			100	mA
		Idle mode			20	$\mu\text{A}$
$T_{START}$	Startup Time	$C_{load} = 2.2\ \mu\text{F}$ , after $V_{DDIN} > 2.7\text{V}$			150	$\mu\text{S}$
$I_O$	Maximum DC Output Current	$V_{DDIN} = 3.3\text{V}$			100	mA
$I_O$	Maximum DC Output Current	$V_{DDIN} = 3.3\text{V}$ , in Idle Mode			1	mA

**Table 85.** Brownout Detector Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V <sub>BOT-</sub>	Threshold Level		1.65	1.68	1.71	V
V <sub>HYST</sub>	Hysteresis	$V_{HYST} = V_{BOT+} - V_{BOT-}$		50	65	mV
I <sub>DD</sub>	Current Consumption	BOD on (GPNVM0 bit active)		12	18	μA
		BOD off (GPNVM0 bit inactive)			1	μA
T <sub>START</sub>	Startup Time			100	200	μs

## Power Consumption

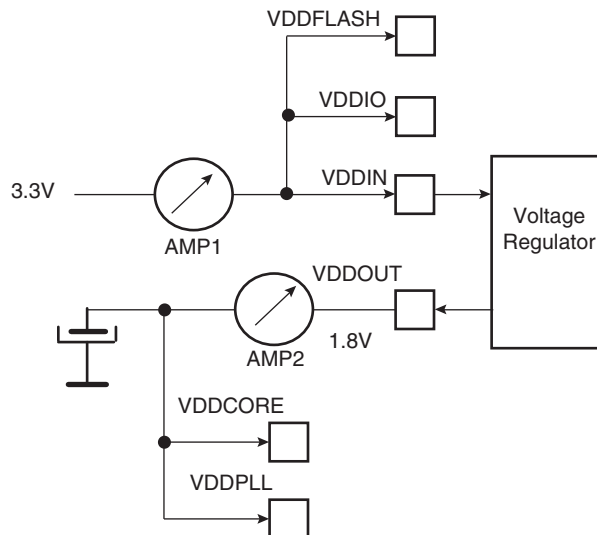
- Typical power consumption of PLLs, Slow Clock and Main Oscillator.
- Power consumption of power supply in two different modes: Active and ultra Low-power.
- Power consumption by peripheral: calculated as the difference in current measurement after having enabled then disabled the corresponding clock.

### Power Consumption Versus Modes

The values in Table 86 and Table 87 on page 410 are estimated values of the power consumption with operating conditions as follows:

- $V_{DDIO} = V_{DDIN} = V_{DDFLASH} = 3.3V$
- $V_{DDCORE} = V_{DDPLL} = 1.85V$
- $T_A = 25^{\circ}C$
- $MCK = 50\text{ MHz}$
- There is no consumption on the I/Os of the device

**Figure 171.** Measure Schematics:





These figures represent the power consumption estimated on the power supplies..

**Table 86.** Power Consumption for Different Modes

Mode	Conditions	Consumption	Unit
Active	Voltage regulator is on. Brown Out Detector is activated. Flash is read. ARM Core clock is 50MHz. Analog-to-Digital Converter activated. All peripheral clocks activated.	onto AMP1	31.3
		onto AMP2	29.3
Ultra low power	Voltage regulator is in Low-power mode. Brown Out Detector is de-activated. Flash is in standby mode. ARM Core clock is 500Hz. Analog-to-Digital Converter de-activated. All peripheral clocks de-activated.	onto AMP1	36.2
		onto AMP2	35.2

## Peripheral Power Consumption in Active Mode

**Table 87.** Power Consumption by Peripheral

Peripheral	Consumption	Unit
PIO Controller	0.4	mA
USART	0.9	
ADC	0.7	
PWM	0.3	
TWI	0.2	
SPI	0.9	
SSC	1.1	
Timer Counter Channels	0.2	

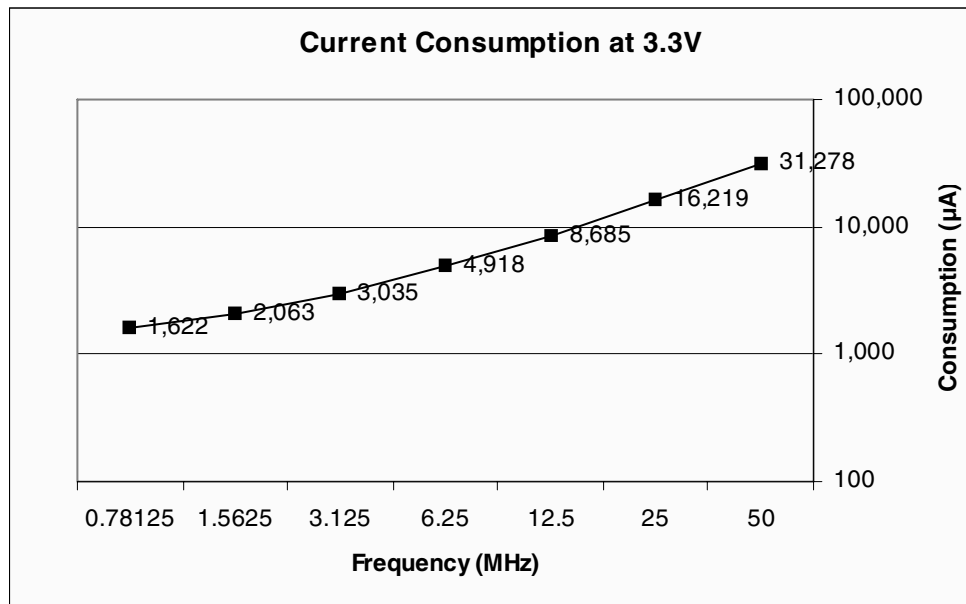
## Power Consumption versus Master Clock Frequency in Active Mode

Figure 172 produces estimated values with operating conditions as follows:

- $V_{DDIO} = V_{DDIN} = V_{DDFLASH} = 3.3V$
- $V_{DDCORE} = V_{DDPLL} = 1.85V$
- $T_A = 25^{\circ}C$
- MCK in the MHz range
- Voltage regulator is on
- Brown-out Detector is activated
- Flash is read
- Analog-to-Digital Converter activated
- All peripheral clocks activated
- There is no consumption on the I/Os of the device

Figure 172 presents the power consumption estimated on the power supply.

**Figure 172.** Power Consumption versus MCK Frequency in Active Mode



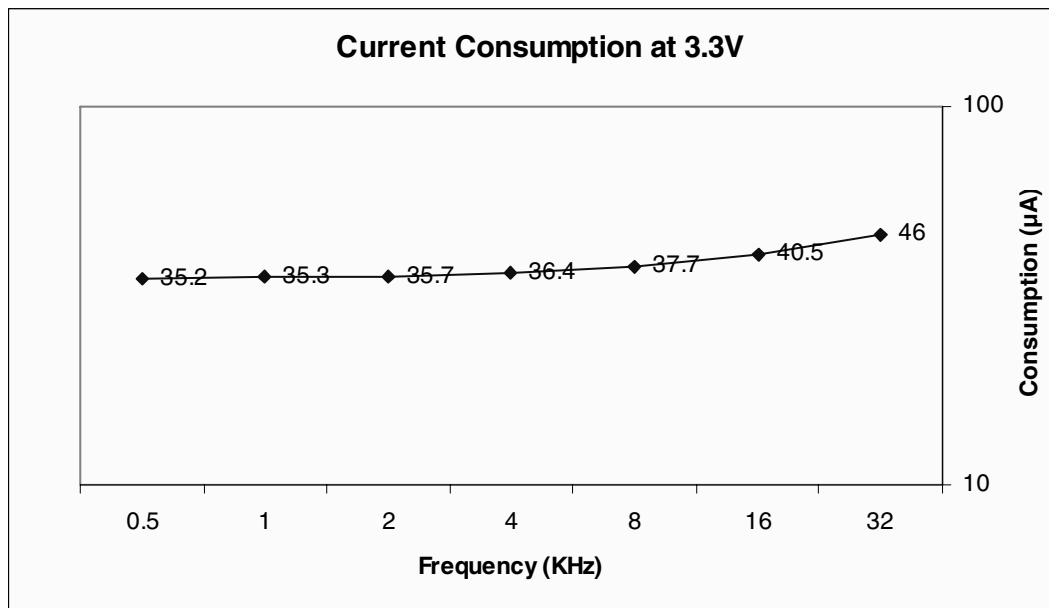
## Power Consumption versus Master Clock Frequency in Ultra Low-power Mode

Figure 173 produces estimated values with operating conditions as follows:

- $V_{DDIO} = V_{DDIN} = V_{DDFLASH} = 3.3V$
- $V_{DDCORE} = V_{DDPLL} = 1.85V$
- $T_A = 25^{\circ}C$
- Voltage regulator is in Low-power mode
- Brown Out Detector is de-activated
- Flash is in standby mode
- Analog-to-digital Converter de-activated
- All peripheral clocks de-activated
- PLL in standby
- Main oscillator in standby
- There is no consumption on the I/Os of the device

Figure 173 presents the power consumption estimated on the power supply.

**Figure 173.** Power Consumption versus MCK Frequency in the Ultra Low Power Mode



## Crystal Oscillators Characteristics

### RC Oscillator Characteristics

**Table 88.** RC Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPRC})$	RC Oscillator Frequency	$V_{DDPLL} = 1.65V$	22	32	42	KHz
	Duty Cycle		45	50	55	%
$t_{ST}$	Startup Time	$V_{DDPLL} = 1.65V$			75	$\mu s$
$I_{OSC}$	Current Consumption	After Startup Time			1.9	$\mu A$

### Main Oscillator Characteristics

**Table 89.** Main Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPMAIN})$	Crystal Oscillator Frequency		3	16	20	MHz
$C_{L1}, C_{L2}$	Internal Load Capacitance ( $C_{L1} = C_{L2}$ )			25		pF
$C_L$	Equivalent Load Capacitance			12.5		pF
	Duty Cycle		40	50	60	%
$t_{ST}$	Startup Time	$V_{DDPLL} = 1.2$ to $2V$ $C_S = 3$ pF <sup>(1)</sup> $1/(t_{CPMAIN}) = 3$ MHz $C_S = 7$ pF <sup>(1)</sup> $1/(t_{CPMAIN}) = 16$ MHz $C_S = 7$ pF <sup>(1)</sup> $1/(t_{CPMAIN}) = 20$ MHz			14.5 1.4 1	ms
$I_{OSC}$	Current Consumption	Active mode			550	$\mu A$
		Standby mode			1	$\mu A$

Notes: 1.  $C_S$  is the shunt capacitance

### XIN Clock Characteristics

**Table 90.** XIN Clock Electrical Characteristics

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPXIN})$	XIN Clock Frequency			50.0	MHz
$t_{CPXIN}$	XIN Clock Period		20.0		ns
$t_{CHXIN}$	XIN Clock High Half-period		$0.4 \times t_{CPXIN}$	$0.6 \times t_{CPXIN}$	
$t_{CLXIN}$	XIN Clock Low Half-period		$0.4 \times t_{CPXIN}$	$0.6 \times t_{CPXIN}$	
$C_{IN}$	XIN Input Capacitance	(1)		25	pF
$R_{IN}$	XIN Pull-down Resistor	(1)		500	k $\Omega$

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 "PMC Clock Generator Main Oscillator Register" on page 166).

## PLL Characteristics

**Table 91.** Phase Lock Loop Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
F <sub>OUT</sub>	Output Frequency	Field out of CKGR_PLL is: 00	80		160	MHz
		10	150		220	MHz
F <sub>IN</sub>	Input Frequency		1		32	MHz
I <sub>PLL</sub>	Current Consumption	Active mode			4	mA
		Standby mode			1	μA

Note: Startup time depends on PLL RC filter. A calculation tool is provided by Atmel.

## ADC Characteristics

**Table 92.** Channel Conversion Time and ADC CLock

Parameter	Conditions	Min	Typ	Max	Units
ADC Clock Frequency				5	MHz
Startup Time	Return from Idle Mode			20	μs
Track and Hold Acquisition Time		600			ns
Conversion Time	ADC Clock = 5 MHz			2	μs
Throughput Rate	ADC Clock = 5 MHz			384	kSPS

**Table 93.** External Voltage Reference Input

Parameter	Conditions	Min	Max	Units
ADVREF Input Voltage Range		2.6	$V_{DDIN}$	V
ADVREF Average Current	On 13 samples with ADC Clock = 5 MHz	12	250	μA

**Table 94.** Analog Inputs

Parameter	Min	Typ	Max	Units
Input Voltage Range	0		$V_{ADVREF}$	
Input Leakage Current		1		μA
Input Capacitance		12	14	pF

**Table 95.** Transfer Characteristics

Parameter	Conditions	Min	Typ	Max	Units
Resolution			10		Bit
Integral Non-linearity				±2	LSB
	ADC Clock = 5 MHz			±3	LSB
Differential Non-linearity				±1	LSB
	ADC Clock = 5 MHz			±2	LSB
Offset Error				±2	LSB
Gain Error				±2	LSB

## AT91SAM7S32 AC Characteristics

### Applicable Conditions and Derating Data

These conditions and derating process apply to the following paragraphs: Clock Characteristics, Embedded Flash Characteristics and JTAG/ICE Timings.

### Conditions and Timings Computation

All delays are given as typical values under the following conditions:

- $V_{DDIO} = 3.3V$
- $V_{DDCORE} = 1.8V$
- Ambient Temperature = 25°C
- Load Capacitance = 0 pF
- The output level change detection is  $(0.5 \times V_{DDIO})$ .
- The input level is 0.8V for a low-level detection and is 2.0V for a high-level detection.

The minimum and maximum values given in the AC characteristics tables of this datasheet take into account process variation and design. In order to obtain the timing for other conditions, the following equation should be used:

$$t = \delta_{T^{\circ}} \times ((\delta_{V_{DDCORE}} \times t_{DATASHEET}) + (\delta_{V_{DDIO}} \times \sum (C_{SIGNAL} \times \delta_{CSIGNAL})))$$

where:

- $\delta_{T^{\circ}}$  is the derating factor in temperature given in Figure 174 on page 417.
- $\delta_{V_{DDCORE}}$  is the derating factor for the Core Power Supply given in Figure 175 on page 417.
- $t_{DATASHEET}$  is the minimum or maximum timing value given in this datasheet for a load capacitance of 0 pF.
- $\delta_{V_{DDIO}}$  is the derating factor for the IO Power Supply given in Figure 176 on page 418.
- $C_{SIGNAL}$  is the capacitance load on the considered output pin<sup>(1)</sup>.
- $\delta_{CSIGNAL}$  is the load derating factor depending on the capacitance load on the related output pins given in Min and Max in this datasheet.

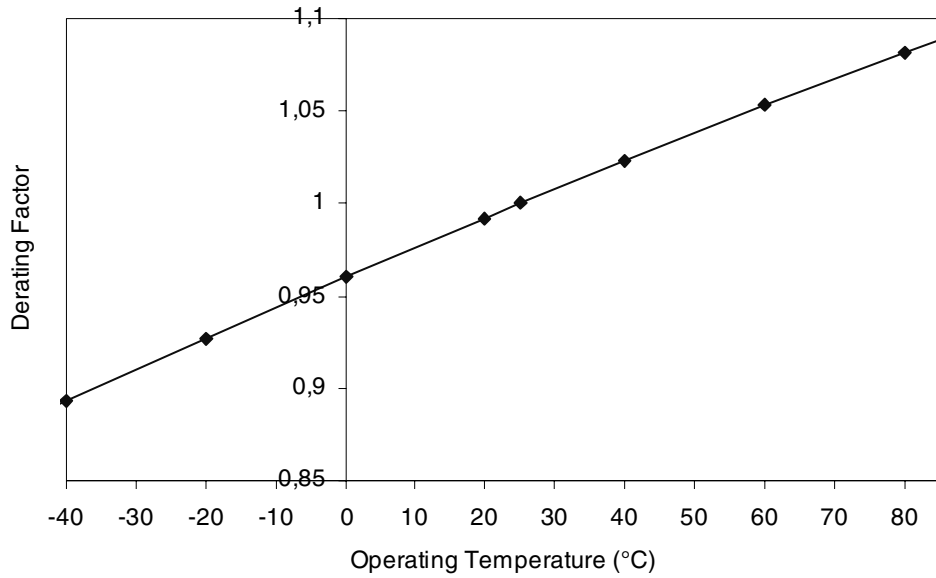
The input delays are given as typical values.

Note: The user must take into account the package capacitance load contribution ( $C_{IN}$ ) described in Table 83, "DC Characteristics," on page 406.



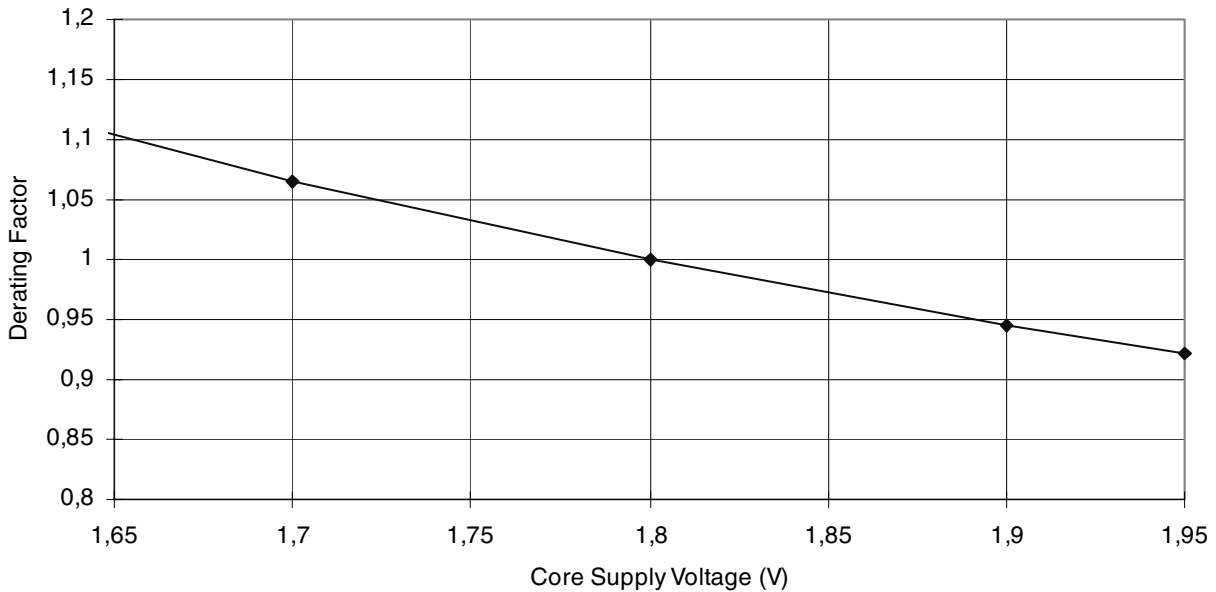
## Temperature Derating Factor

Figure 174. Derating Curve for Different Operating Temperatures



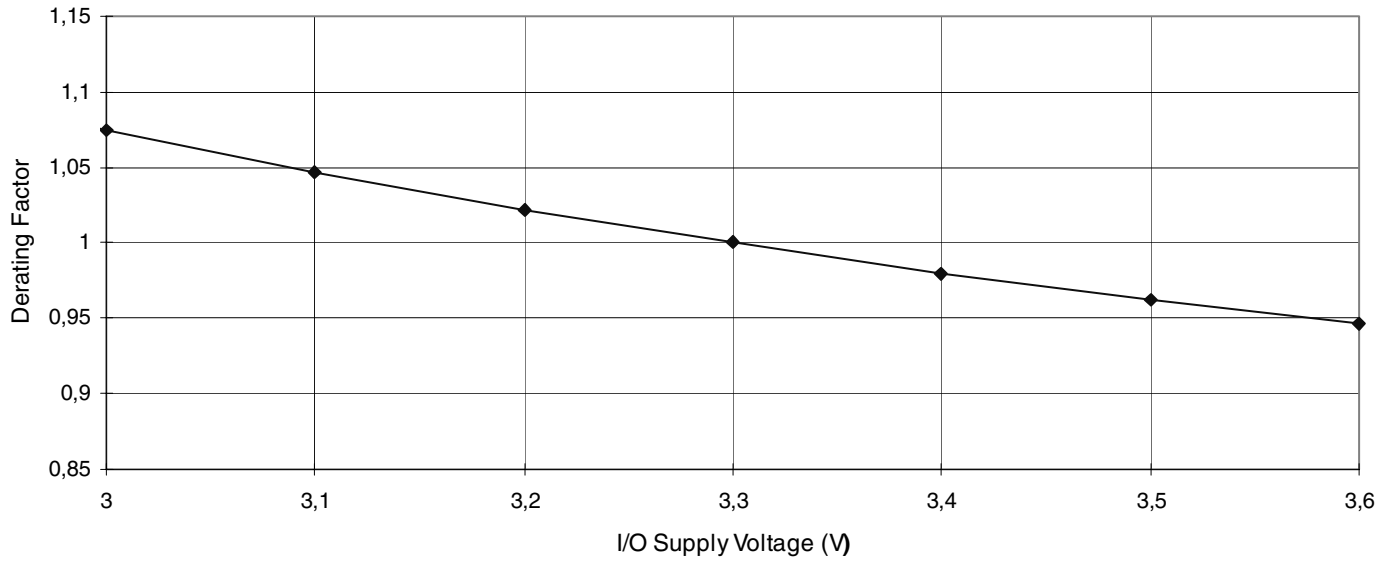
## V<sub>DDCORE</sub> Voltage Derating Factor

Figure 175. Derating Curve for Different Core Supply Voltages



## V<sub>DDIO</sub> Voltage Derating Factor

**Figure 176.** Derating Curve for Different IO Supply Voltages



Note: The derating factor in this example is applicable only to timings related to output pins.

## Clock Characteristics

These parameters are given in the following conditions:

- $V_{DDCORE} = 1.8V$
- Ambient Temperature = 25°C

The Temperature Derating Factor described in “AT91SAM7S32 AC Characteristics” on page 416 and “VDDCORE Voltage Derating Factor” on page 417 are both applicable to these characteristics.

## Master Clock Characteristics

**Table 96.** Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPMCK})$	Master Clock Frequency			73	MHz

## Embedded Flash Characteristics

**Table 97.** DC Flash Characteristics

Symbol	Parameter	Conditions	Min	Max	Units
$T_{PU}$	Power-up delay			30	$\mu$ S
$I_{SB}$	Standby current	@85°C onto VDDCORE = 1.8V onto VDDFLASH = 3.3V		0 30	$\mu$ A
$I_{CC}$	Active current	Random Read @ 40MHz onto VDDCORE = 1.8V onto VDDFLASH = 3.3V		5.0 1.0	mA
		Write onto VDDCORE = 1.8V onto VDDFLASH = 3.3V		500 8.0	$\mu$ A mA

The maximum operating frequency is given in Table 97 but is limited by the Embedded Flash access time when the processor is fetching code out of it. Table 98 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 98.** Embedded Flash Wait States

FWS	Read Operations	Maximum Operating Frequency (MHz)
0	1 cycle	40
1	2 cycles	$1/(t_{CPMCK})$
2	3 cycles	$1/(t_{CPMCK})$
3	4 cycles	$1/(t_{CPMCK})$

**Table 99.** AC Flash Characteristics

Parameter	Conditions	Min	Max	Units
Program Cycle Time	per page including auto-erase		4	ms
	per page without auto-erase		2	ms
Full Chip Erase		10		ms

## JTAG/ICE Timings

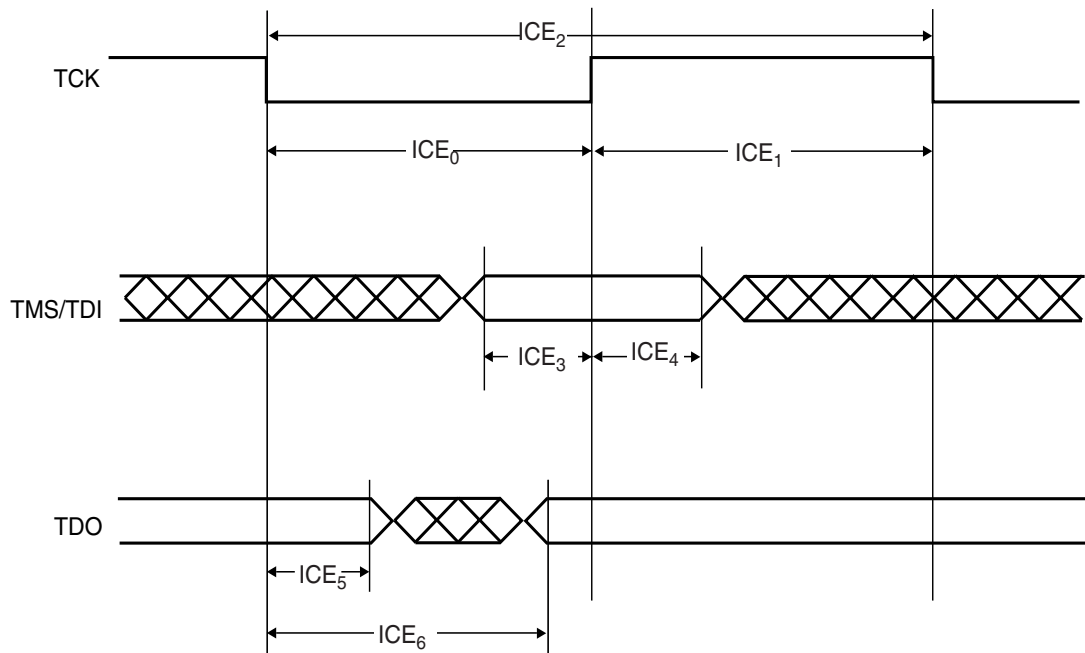
### ICE Interface Signals

Table 100 shows timings relative to operating condition limits defined in the section “Conditions and Timings Computation” on page 416.

**Table 100.** ICE Interface Timing Specification

Symbol	Parameter	Conditions	Min	Max	Units
ICE <sub>0</sub>	TCK Low Half-period		51		ns
ICE <sub>1</sub>	TCK High Half-period		51		ns
ICE <sub>2</sub>	TCK Period		102		ns
ICE <sub>3</sub>	TDI, TMS, Setup before TCK High		3		ns
ICE <sub>4</sub>	TDI, TMS, Hold after TCK High		0		ns
ICE <sub>5</sub>	TDO Hold Time	C <sub>TDO</sub> = 0 pF	3		ns
		C <sub>TDO</sub> derating	0.037		ns/pF
ICE <sub>6</sub>	TCK Low to TDO Valid	C <sub>TDO</sub> = 0 pF		13	ns
		C <sub>TDO</sub> derating		0.037	ns/pF

**Figure 177.** ICE Interface Signals



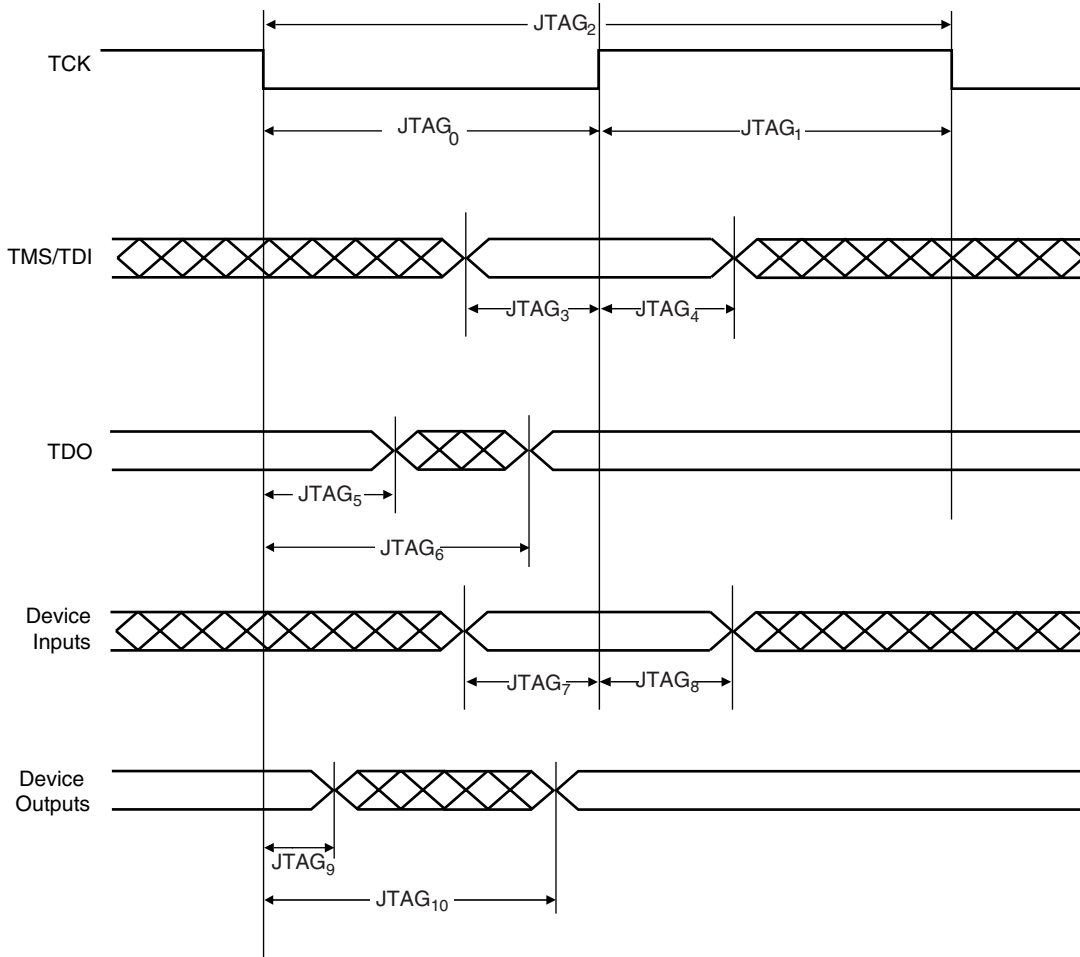
## JTAG Interface Signals

The following table shows timings relative to operating condition limits defined in the section “Conditions and Timings Computation” on page 416.

**Table 101.** JTAG Interface Timing specification

Symbol	Parameter	Conditions	Min	Max	Units
JTAG <sub>0</sub>	TCK Low Half-period		6.5		ns
JTAG <sub>1</sub>	TCK High Half-period		5.5		ns
JTAG <sub>2</sub>	TCK Period		12		ns
JTAG <sub>3</sub>	TDI, TMS Setup before TCK High		2		ns
JTAG <sub>4</sub>	TDI, TMS Hold after TCK High		3		ns
JTAG <sub>5</sub>	TDO Hold Time	C <sub>TDO</sub> = 0 pF	2		ns
		C <sub>TDO</sub> derating	0.037		ns/pF
JTAG <sub>6</sub>	TCK Low to TDO Valid	C <sub>TDO</sub> = 0 pF		15	ns
		C <sub>TDO</sub> derating		0.037	ns/pF
JTAG <sub>7</sub>	Device Inputs Setup Time		0		ns
JTAG <sub>8</sub>	Device Inputs Hold Time		3		ns
JTAG <sub>9</sub>	Device Outputs Hold Time	C <sub>OUT</sub> = 0 pF	4		ns
		C <sub>OUT</sub> derating	0.037		ns/pF
JTAG <sub>10</sub>	TCK to Device Outputs Valid	C <sub>OUT</sub> = 0 pF		20	ns
		C <sub>OUT</sub> derating		0.037	ns/pF

**Figure 178. JTAG Interface Signals**



# AT91SAM7S32 Mechanical Characteristics

## Thermal Considerations

### Thermal Data

In Table 102, the device lifetime is estimated using the MIL-217 standard in the “moderately controlled” environmental model (this model is described as corresponding to an installation in a permanent rack with adequate cooling air), depending on the device Junction Temperature. (For details see the section “Junction Temperature” on page 425.)

Note that the user must be extremely cautious with this MTBF calculation. It should be noted that the MIL-217 model is pessimistic with respect to observed values due to the way the data/models are obtained (test under severe conditions). The life test results that have been measured are always better than the predicted ones.

**Table 102.** MTBF Versus Junction Temperature

Junction Temperature ( $T_J$ ) (°C)	Estimated Lifetime (MTBF) (Year)
100	17.5
125	9
150	5
175	3

Table 103 summarizes the thermal resistance data depending on the package.

**Table 103.** Thermal Resistance Data

Symbol	Parameter	Condition	Package	Typ	Unit
$\theta_{JA}$	Junction-to-ambient thermal resistance	Still Air	LQFP48	49.5	°C/W
$\theta_{JC}$	Junction-to-case thermal resistance		LQFP48	12.7	



## Junction Temperature

The average chip-junction temperature,  $T_J$ , in °C can be obtained from the following:

7.  $T_J = T_A + (P_D \times \theta_{JA})$
8.  $T_J = T_A + (P_D \times (\theta_{HEATSINK} + \theta_{JC}))$

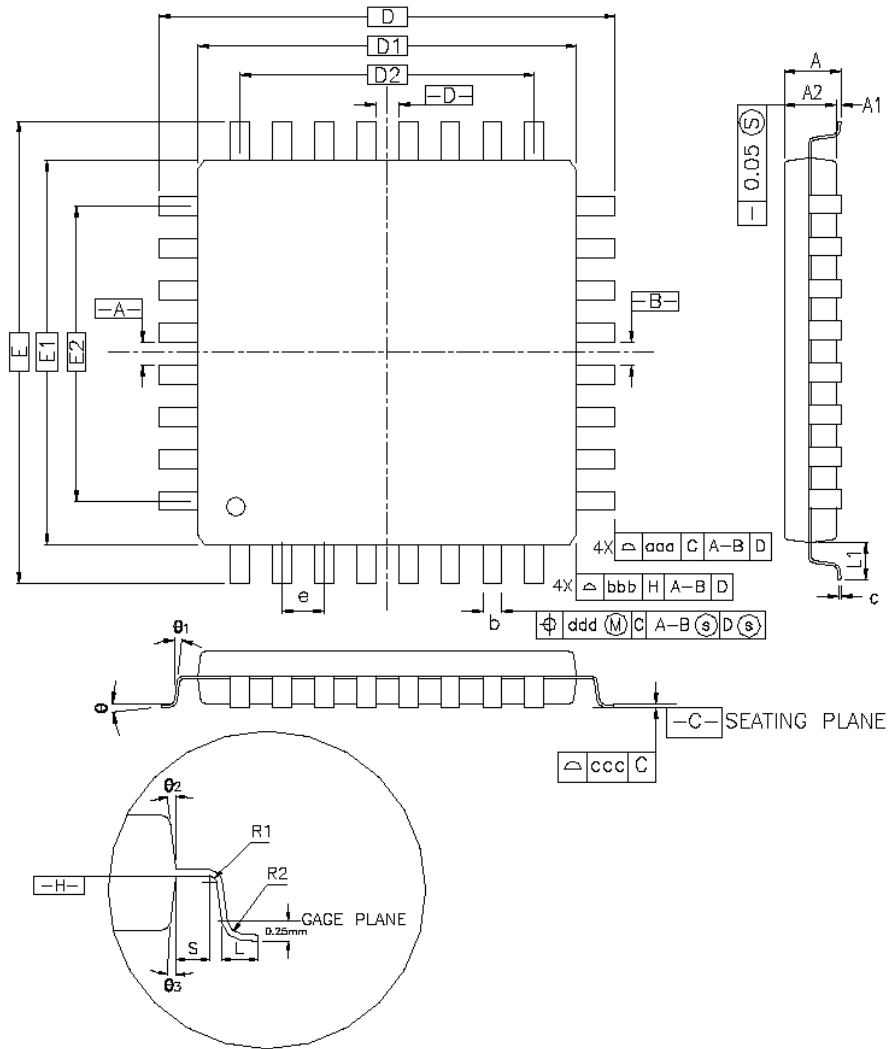
where:

- $\theta_{JA}$  = package thermal resistance, Junction-to-ambient (°C/W), provided in Table 103 on page 424.
- $\theta_{JC}$  = package thermal resistance, Junction-to-case thermal resistance (°C/W), provided in Table 103 on page 424.
- $\theta_{HEATSINK}$  = 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 408.
- $T_A$  = ambient temperature (°C).

From the first equation, the user can derive the estimated lifetime of the chip and decide if a cooling device is necessary or not. If a cooling device is to be fitted on the chip, the second equation should be used to compute the resulting average chip-junction temperature  $T_J$  in °C.

# Package Drawings

Figure 179. 48-lead LQFP Package Drawing



**Table 104.** 48-lead LQFP Package Dimensions (in mm)

CONTROL DIMENSIONS ARE IN MILLIMETERS.

SYMBOL	MILLIMETER			INCH		
	MIN.	NOM.	MAX.	MIN.	NOM.	MAX.
A	—	—	1.60	—	—	0.063
A1	0.05	—	0.15	0.002	—	0.006
A2	1.35	1.40	1.45	0.053	0.055	0.057
D	9.00 BSC.			0.354 BSC.		
D1	7.00 BSC.			0.276 BSC.		
E	9.00 BSC.			0.354 BSC.		
E1	7.00 BSC.			0.276 BSC.		
R2	0.08	—	0.20	0.003	—	0.008
R1	0.08	—	—	0.003	—	—
Ø	0°	3.5°	7°	0°	3.5°	7°
Ø <sub>1</sub>	0°	—	—	0°	—	—
Ø <sub>2</sub>	11°	12°	13°	11°	12°	13°
Ø <sub>3</sub>	11°	12°	13°	11°	12°	13°
c	0.09	—	0.20	0.004	—	0.008
L	0.45	0.60	0.75	0.018	0.024	0.030
L <sub>1</sub>	1.00 REF			0.039 REF		
S	0.20	—	—	0.008	—	—
b	0.17	0.20	0.27	0.007	0.008	0.011
e	0.50 BSC.			0.020 BSC.		
D2	5.50			0.217		
E2	5.50			0.217		
TOLERANCES OF FORM AND POSITION						
aaa	0.20			0.008		
bbb	0.20			0.008		
ccc	0.08			0.003		
ddd	0.08			0.003		

**Table 105.** Device and 48-lead LQFP Package Maximum Weight

700	mg
-----	----

**Table 106.** 48-lead LQFP Package Characteristics

Moisture Sensitivity Level	3
----------------------------	---

## Soldering Profile

Table 107 gives the recommended soldering profile from J-STD-20.

**Table 107.** Soldering Profile

	Convection or IR/Convection	VPR
Average Ramp-up Rate (183°C to Peak)	3°C/sec. max.	10°C/sec.
Preheat Temperature 125°C ±25°C	120 sec. max	
Temperature Maintained Above 183°C	60 sec. to 150 sec.	
Time within 5°C of Actual Peak Temperature	10 sec. to 20 sec.	60 sec.
Peak Temperature Range	220 +5/-0°C or 235 +5/-0°C	215 to 219°C or 235 +5/-0°C
Ramp-down Rate	6°C/sec.	10°C/sec.
Time 25°C to Peak Temperature	6 min. max	

Small packages may be subject to higher temperatures if they are reflowed in boards with larger components. In this case, small packages may have to withstand temperatures of up to 235°C, not 220°C (IR reflow).

Recommended package reflow conditions depend on package thickness and volume. See Table 108.

**Table 108.** Recommended Package Reflow Conditions <sup>(1, 2, 3)</sup>

Parameter	Temperature
Convection	235 +5/-0°C
VPR	235 +5/-0°C
IR/Convection	235 +5/-0°C

When certain small thin packages are used on boards without larger packages, these small packages may be classified at 220°C instead of 235°C.

- Notes:
1. The packages are qualified by Atmel by using IR reflow conditions, not convection or VPR.
  2. By default, the package level 1 is qualified at 220°C (unless 235°C is stipulated).
  3. The body temperature is the most important parameter but other profile parameters such as total exposure time to hot temperature or heating rate may also influence component reliability.

A maximum of three reflow passes is allowed per component.

## AT91SAM7S32 Ordering Information

Table 109. Ordering Information

Ordering Code	Package	Temperature Operating Range
AT91SAM7S32-AI	LQFP 48	Industrial (-40°C to 85°C)



## Table of Contents

<b>Features</b> .....	<b>1</b>
<b>Description</b> .....	<b>2</b>
<b>Block Diagram</b> .....	<b>3</b>
Signal Description .....	4
Package and Pinout .....	7
48-lead LQFP Mechanical Overview.....	7
Pinout.....	7
<b>Power Considerations</b> .....	<b>8</b>
Power Supplies .....	8
Power Consumption.....	8
Voltage Regulator .....	8
Typical Powering Schematics .....	9
<b>I/O Lines Considerations</b> .....	<b>10</b>
JTAG Port Pins .....	10
Test Pin .....	10
Reset Pin.....	10
ERASE Pin.....	10
PIO Controller Lines .....	10
I/O Line Drive Levels.....	10
<b>Processor and Architecture</b> .....	<b>11</b>
ARM7TDMI Processor .....	11
Debug and Test Features .....	11
Memory Controller.....	11
Peripheral Data Controller.....	12
<b>Memories</b> .....	<b>13</b>
Memory Mapping .....	13
Embedded Flash .....	14
Fast Flash Programming Interface.....	15
<b>System Controller</b> .....	<b>16</b>
System Controller Mapping.....	17
Reset Controller .....	18
Clock Generator.....	19
Power Management Controller .....	20
Advanced Interrupt Controller .....	20
Debug Unit .....	21
Periodic Interval Timer .....	21
Watchdog Timer.....	21
Real-time Timer.....	21
PIO Controller .....	21
Voltage Regulator Controller.....	21
<b>Peripherals</b> .....	<b>22</b>
Peripheral Mapping .....	22
Peripheral Multiplexing on PIO Lines .....	23
Peripheral Identifiers .....	24
Serial Peripheral Interface.....	24
Two-wire Interface.....	25





USART .....	25
Serial Synchronous Controller .....	25
Timer Counter .....	25
PWM Controller .....	26
Analog-to-digital Converter .....	26

---

<b>ARM7TDMI Processor Overview .....</b>	<b>27</b>
<b>Overview .....</b>	<b>27</b>
<b>ARM7TDMI Processor .....</b>	<b>28</b>
Instruction Type .....	28
Data Type .....	28
ARM7TDMI Operating Mode .....	28
ARM7TDMI Registers .....	28
ARM Instruction Set Overview .....	30
Thumb Instruction Set Overview .....	31

---

<b>AT91SAM7S32 Debug and Test Features .....</b>	<b>33</b>
<b>Description .....</b>	<b>33</b>
<b>Block Diagram .....</b>	<b>33</b>
<b>Application Examples .....</b>	<b>34</b>
Debug Environment .....	34
Test Environment .....	35
<b>Debug and Test Pin Description .....</b>	<b>35</b>
<b>Functional Description .....</b>	<b>36</b>
<b>Test Pin .....</b>	<b>36</b>
Embedded In-circuit Emulator .....	36
Debug Unit .....	36
IEEE 1149.1 JTAG Boundary Scan .....	36
ID Code Register .....	40

---

<b>Reset Controller (RSTC) .....</b>	<b>41</b>
<b>Overview .....</b>	<b>41</b>
<b>Block Diagram .....</b>	<b>41</b>
<b>Functional Description .....</b>	<b>42</b>
NRST Manager .....	42
Brownout Manager .....	43
Reset States .....	44
Reset State Priorities .....	49
Reset Controller Status Register .....	49
<b>Reset Controller (RSTC) User Interface .....</b>	<b>51</b>
Reset Controller Control Register .....	52
Reset Controller Status Register .....	53
Reset Controller Mode Register .....	54



<b>Real-time Timer (RTT)</b> .....	<b>55</b>
<b>Overview</b> .....	<b>55</b>
<b>Block Diagram</b> .....	<b>55</b>
<b>Functional Description</b> .....	<b>56</b>
<b>Real-time Timer (RTT) User Interface</b> .....	<b>58</b>
Real-time Timer Mode Register .....	59
Real-time Timer Alarm Register.....	60
Real-time Timer Value Register .....	61
Real-time Timer Status Register .....	62
<hr/>	
<b>Periodic Interval Timer (PIT)</b> .....	<b>63</b>
<b>Overview</b> .....	<b>63</b>
<b>Block Diagram</b> .....	<b>63</b>
<b>Functional Description</b> .....	<b>64</b>
<b>Periodic Interval Timer (PIT) User Interface</b> .....	<b>66</b>
Periodic Interval Timer Mode Register .....	67
Periodic Interval Timer Status Register.....	68
Periodic Interval Timer Value Register.....	69
Periodic Interval Timer Image Register .....	70
<hr/>	
<b>Watchdog Timer (WDT)</b> .....	<b>71</b>
<b>Overview</b> .....	<b>71</b>
<b>Block Diagram</b> .....	<b>71</b>
<b>Functional Description</b> .....	<b>72</b>
<b>Watchdog Timer (WDT) User Interface</b> .....	<b>74</b>
Watchdog Timer Control Register .....	75
Watchdog Timer Mode Register .....	76
Watchdog Timer Status Register .....	77
<hr/>	
<b>Voltage Regulator Mode Controller (VREG)</b> .....	<b>79</b>
<b>Overview</b> .....	<b>79</b>
<b>Voltage Regulator Power Controller (VREG) User Interface</b> .....	<b>79</b>
Voltage Regulator Mode Register .....	79
<hr/>	
<b>Memory Controller (MC)</b> .....	<b>81</b>
<b>Overview</b> .....	<b>81</b>
<b>Block Diagram</b> .....	<b>81</b>
<b>Functional Description</b> .....	<b>82</b>
Bus Arbiter .....	82
Address Decoder .....	82
Remap Command .....	83
Abort Status .....	84
Embedded Flash Controller .....	84
Misalignment Detector .....	84





<b>Memory Controller (MC) User Interface</b> .....	<b>85</b>
MC Remap Control Register .....	86
MC Abort Status Register .....	87
MC Abort Address Status Register .....	88

---

<b>Embedded Flash Controller (EFC)</b> .....	<b>89</b>
<b>Overview</b> .....	<b>89</b>
<b>Functional Description</b> .....	<b>89</b>
Embedded Flash Organization.....	89
Read Operations .....	91
Write Operations .....	93
Flash Commands .....	93
<b>Embedded Flash Controller (EFC) User Interface</b> .....	<b>98</b>
MC Flash Mode Register .....	99
MC Flash Command Register .....	100
MC Flash Status Register .....	102

---

<b>Fast Flash Programming Interface (FFPI)</b> .....	<b>103</b>
<b>Overview</b> .....	<b>103</b>
<b>Parallel Fast Flash Programming</b> .....	<b>104</b>
Device Configuration.....	104
Signal Names.....	105
Entering Programming Mode .....	106
Read Handshaking.....	107
Device Operations.....	108
<b>Serial Fast Flash Programming</b> .....	<b>112</b>
Device Configuration.....	112
Entering Serial Programming Mode .....	113
Read/Write Handshake .....	113
Device Operations.....	114

---

<b>Peripheral Data Controller (PDC)</b> .....	<b>117</b>
<b>Overview</b> .....	<b>117</b>
<b>Block Diagram</b> .....	<b>117</b>
<b>Functional Description</b> .....	<b>118</b>
Configuration.....	118
Memory Pointers .....	118
Transfer Counters .....	118
Data Transfers .....	119
Priority of PDC Transfer Requests.....	119
<b>Peripheral Data Controller (PDC) User Interface</b> .....	<b>120</b>
PDC Receive Pointer Register.....	121
PDC Receive Counter Register .....	121
PDC Transmit Pointer Register.....	122
PDC Transmit Counter Register .....	122

PDC Receive Next Pointer Register .....	123
PDC Receive Next Counter Register .....	123
PDC Transmit Next Pointer Register .....	124
PDC Transmit Next Counter Register .....	124
PDC Transfer Control Register .....	125
PDC Transfer Status Register.....	126

---

<b>Advanced Interrupt Controller (AIC).....</b>	<b>127</b>
<b>Overview.....</b>	<b>127</b>
<b>Block Diagram.....</b>	<b>127</b>
<b>Application Block Diagram .....</b>	<b>127</b>
<b>AIC Detailed Block Diagram .....</b>	<b>128</b>
<b>I/O Line Description.....</b>	<b>128</b>
<b>Product Dependencies.....</b>	<b>128</b>
I/O Lines.....	128
Power Management.....	128
Interrupt Sources.....	128
<b>Functional Description.....</b>	<b>130</b>
Interrupt Source Control.....	130
Interrupt Latencies .....	132
Normal Interrupt .....	133
Fast Interrupt.....	135
Protect Mode.....	138
Spurious Interrupt.....	139
General Interrupt Mask .....	139
<b>Advanced Interrupt Controller (AIC) User Interface.....</b>	<b>140</b>
Base Address.....	140
AIC Source Mode Register .....	141
AIC Source Vector Register .....	142
AIC Interrupt Vector Register .....	142
AIC FIQ Vector Register .....	143
AIC Interrupt Status Register .....	144
AIC Interrupt Pending Register .....	144
AIC Interrupt Mask Register.....	145
AIC Core Interrupt Status Register .....	145
AIC Interrupt Enable Command Register.....	146
AIC Interrupt Disable Command Register.....	146
AIC Interrupt Clear Command Register .....	147
AIC Interrupt Set Command Register .....	147
AIC End of Interrupt Command Register .....	148
AIC Spurious Interrupt Vector Register.....	148
AIC Debug Control Register.....	149
AIC Fast Forcing Enable Register.....	149
AIC Fast Forcing Disable Register.....	150
AIC Fast Forcing Status Register.....	150



<b>Clock Generator.....</b>	<b>151</b>
<b>Description .....</b>	<b>151</b>
Slow Clock RC Oscillator .....	151
Main Oscillator .....	151
Divider and PLL Block.....	153
<hr/>	
<b>Power Management Controller (PMC) .....</b>	<b>154</b>
Description .....	154
Master Clock Controller.....	154
Processor Clock Controller .....	154
Peripheral Clock Controller .....	155
Programmable Clock Output Controller .....	155
Programming Sequence .....	155
Clock Switching Details.....	158
Power Management Controller (PMC) User Interface .....	161
<hr/>	
<b>Debug Unit (DBGU) .....</b>	<b>173</b>
<b>Overview.....</b>	<b>173</b>
<b>Block Diagram.....</b>	<b>174</b>
<b>Product Dependencies.....</b>	<b>175</b>
I/O Lines.....	175
Power Management .....	175
Interrupt Source .....	175
<b>UART Operations.....</b>	<b>175</b>
Baud Rate Generator .....	175
Receiver .....	176
Transmitter.....	178
Peripheral Data Controller.....	179
Test Modes .....	179
Debug Communication Channel Support.....	180
Chip Identifier .....	181
ICE Access Prevention .....	181
<b>Debug Unit User Interface .....</b>	<b>182</b>
Debug Unit Control Register .....	183
Debug Unit Mode Register .....	184
Debug Unit Interrupt Enable Register .....	185
Debug Unit Interrupt Disable Register .....	186
Debug Unit Interrupt Mask Register.....	187
Debug Unit Status Register.....	188
Debug Unit Receiver Holding Register .....	190
Debug Unit Transmit Holding Register.....	191
Debug Unit Baud Rate Generator Register.....	191
Debug Unit Chip ID Register.....	192
Debug Unit Chip ID Extension Register .....	195

Debug Unit Force NTRST Register.....	195
--------------------------------------	-----

---

<b>Parallel Input/Output Controller (PIO) .....</b>	<b>197</b>
<b>Overview.....</b>	<b>197</b>
<b>Block Diagram.....</b>	<b>198</b>
<b>Application Block Diagram.....</b>	<b>198</b>
<b>Product Dependencies.....</b>	<b>199</b>
Pin Multiplexing.....	199
External Interrupt Lines.....	199
Power Management.....	199
Interrupt Generation.....	199
<b>Functional Description.....</b>	<b>200</b>
Pull-up Resistor Control.....	200
I/O Line or Peripheral Function Selection.....	201
Peripheral A or B Selection.....	201
Output Control.....	201
Synchronous Data Output.....	202
Multi Drive Control (Open Drain).....	202
Output Line Timings.....	202
Inputs.....	202
Input Glitch Filtering.....	203
Input Change Interrupt.....	203
<b>I/O Lines Programming Example.....</b>	<b>205</b>
<b>Parallel Input/Output Controller (PIO) User Interface.....</b>	<b>206</b>
PIO Controller PIO Enable Register.....	208
PIO Controller PIO Disable Register.....	208
PIO Controller PIO Status Register.....	209
PIO Controller Output Enable Register.....	209
PIO Controller Output Disable Register.....	210
PIO Controller Output Status Register.....	210
PIO Controller Input Filter Enable Register.....	211
PIO Controller Input Filter Disable Register.....	211
PIO Controller Input Filter Status Register.....	212
PIO Controller Set Output Data Register.....	212
PIO Controller Clear Output Data Register.....	213
PIO Controller Output Data Status Register.....	213
PIO Controller Pin Data Status Register.....	214
PIO Controller Interrupt Enable Register.....	214
PIO Controller Interrupt Disable Register.....	215
PIO Controller Interrupt Mask Register.....	215
PIO Controller Interrupt Status Register.....	216
PIO Multi-driver Enable Register.....	216
PIO Multi-driver Disable Register.....	217
PIO Multi-driver Status Register.....	217
PIO Pull Up Disable Register.....	218
PIO Pull Up Enable Register.....	218





PIO Pull Up Status Register .....	219
PIO Peripheral A Select Register .....	219
PIO Peripheral B Select Register .....	220
PIO Peripheral A B Status Register .....	220
PIO Output Write Enable Register .....	221
PIO Output Write Disable Register .....	221
PIO Output Write Status Register .....	222

---

<b>Serial Peripheral Interface (SPI) .....</b>	<b>223</b>
<b>Overview .....</b>	<b>223</b>
<b>Block Diagram .....</b>	<b>224</b>
<b>Application Block Diagram .....</b>	<b>224</b>
<b>Signal Description .....</b>	<b>225</b>
<b>Product Dependencies .....</b>	<b>225</b>
I/O Lines .....	225
Power Management .....	225
Interrupt .....	225
<b>Functional Description .....</b>	<b>226</b>
Modes of Operation .....	226
Data Transfer .....	226
Master Mode Operations .....	228
SPI Slave Mode .....	233
<b>Serial Peripheral Interface (SPI) User Interface .....</b>	<b>235</b>
SPI Control Register .....	236
SPI Mode Register .....	237
SPI Receive Data Register .....	239
SPI Transmit Data Register .....	240
SPI Status Register .....	241
SPI Interrupt Enable Register .....	243
SPI Interrupt Disable Register .....	244
SPI Interrupt Mask Register .....	245
SPI Chip Select Register .....	246

---

<b>Two-wire Interface (TWI) .....</b>	<b>249</b>
<b>Overview .....</b>	<b>249</b>
<b>Block Diagram .....</b>	<b>249</b>
<b>Application Block Diagram .....</b>	<b>249</b>
<b>Product Dependencies .....</b>	<b>250</b>
I/O Lines Description .....	250
Power Management .....	250
Interrupt .....	250
<b>Functional Description .....</b>	<b>251</b>
Transfer Format .....	251
Modes of Operation .....	251
Transmitting Data .....	251

Read/Write Flowcharts.....	254
<b>Two-wire Interface (TWI) User Interface .....</b>	<b>256</b>
TWI Control Register.....	257
TWI Master Mode Register .....	258
TWI Internal Address Register .....	259
TWI Clock Waveform Generator Register.....	259
TWI Status Register .....	260
TWI Interrupt Enable Register.....	261
TWI Interrupt Disable Register.....	262
TWI Interrupt Mask Register .....	263
TWI Receive Holding Register .....	264
TWI Transmit Holding Register .....	264

---

## **Universal Synchronous/Asynchronous Receiver/Transmitter (USART) 265**

<b>Overview.....</b>	<b>265</b>
<b>Block Diagram.....</b>	<b>266</b>
<b>Application Block Diagram .....</b>	<b>267</b>
<b>I/O Lines Description .....</b>	<b>267</b>
<b>Product Dependencies.....</b>	<b>268</b>
I/O Lines.....	268
Power Management .....	268
Interrupt.....	268
<b>Functional Description.....</b>	<b>269</b>
Baud Rate Generator .....	269
Receiver and Transmitter Control .....	273
Synchronous and Asynchronous Modes.....	273
ISO7816 Mode .....	283
IrDA Mode .....	285
RS485 Mode .....	288
Modem Mode .....	289
Test Modes .....	290
<b>USART User Interface .....</b>	<b>292</b>
USART Control Register .....	293
USART Mode Register.....	295
USART Interrupt Enable Register .....	298
USART Interrupt Disable Register .....	299
USART Interrupt Mask Register.....	300
USART Channel Status Register .....	301
USART Receive Holding Register .....	303
USART Transmit Holding Register .....	303
USART Baud Rate Generator Register .....	304
USART Receiver Time-out Register .....	305
USART Transmitter Timeguard Register .....	306
USART FI DI RATIO Register .....	307
USART Number of Errors Register .....	308
USART IrDA FILTER Register .....	308





---

<b>Synchronous Serial Controller (SSC)</b> .....	<b>309</b>
<b>Overview</b> .....	<b>309</b>
<b>Block Diagram</b> .....	<b>309</b>
<b>Application Block Diagram</b> .....	<b>310</b>
<b>Pin Name List</b> .....	<b>311</b>
<b>Product Dependencies</b> .....	<b>311</b>
I/O Lines.....	311
Power Management .....	311
Interrupt.....	311
<b>Functional Description</b> .....	<b>311</b>
Clock Management .....	312
Clock Divider.....	313
Transmitter Operations .....	315
Receiver Operations .....	316
Start.....	316
Frame Sync.....	318
Data Format .....	318
Loop Mode .....	320
Interrupt.....	320
<b>SSC Application Examples</b> .....	<b>322</b>
<b>Synchronous Serial Controller (SSC) User Interface</b> .....	<b>324</b>
SSC Control Register.....	325
SSC Clock Mode Register .....	325
SSC Receive Clock Mode Register .....	326
SSC Receive Frame Mode Register .....	328
SSC Transmit Clock Mode Register .....	330
SSC Transmit Frame Mode Register .....	332
SSC Receive Holding Register .....	334
SSC Transmit Holding Register .....	334
SSC Receive Synchronization Holding Register.....	335
SSC Transmit Synchronization Holding Register.....	335
SSC Status Register .....	336
SSC Interrupt Enable Register.....	338
SSC Interrupt Disable Register .....	339
SSC Interrupt Mask Register .....	340

---

<b>Timer/Counter (TC)</b> .....	<b>341</b>
<b>Overview</b> .....	<b>341</b>
<b>Block Diagram</b> .....	<b>341</b>
<b>Pin Name List</b> .....	<b>342</b>
<b>Product Dependencies</b> .....	<b>342</b>
I/O Lines.....	342
Power Management .....	342
Interrupt.....	342
<b>Functional Description</b> .....	<b>342</b>



TC Description .....	342
Capture Operating Mode.....	345
Waveform Operating Mode .....	347
<b>Timer/Counter (TC) User Interface .....</b>	<b>354</b>
Global Register Mapping .....	354
Channel Memory Mapping .....	354
TC Block Control Register.....	355
TC Block Mode Register .....	355
TC Channel Control Register .....	356
TC Channel Mode Register: Capture Mode .....	357
TC Channel Mode Register: Waveform Mode .....	359
TC Counter Value Register .....	362
TC Register A.....	362
TC Register B.....	363
TC Register C .....	363
TC Status Register .....	364
TC Interrupt Enable Register .....	366
TC Interrupt Disable Register.....	367
TC Interrupt Mask Register .....	368

---

<b>Pulse Width Modulation Controller (PWM) .....</b>	<b>369</b>
<b>Overview.....</b>	<b>369</b>
<b>Block Diagram.....</b>	<b>369</b>
<b>I/O Lines Description.....</b>	<b>370</b>
<b>Product Dependencies.....</b>	<b>370</b>
I/O Lines.....	370
Power Management .....	370
Interrupt Sources.....	370
<b>Functional Description.....</b>	<b>371</b>
PWM Clock Generator .....	371
PWM Channel.....	372
PWM Controller Operations .....	376
<b>PWM User Interface.....</b>	<b>378</b>
PWM Register Mapping .....	378
PWM Mode Register .....	379
PWM Enable Register.....	380
PWM Disable Register .....	380
PWM Status Register.....	381
PWM Interrupt Enable Register .....	382
PWM Interrupt Disable Register.....	382
PWM Interrupt Mask Register .....	383
PWM Interrupt Status Register .....	383
PWM Channel Mode Register.....	384
PWM Channel Duty Cycle Register .....	385
PWM Channel Period Register .....	386
PWM Channel Counter Register.....	387





PWM Channel Update Register .....	387
PWM Version Register .....	388

---

<b>Analog-to-digital Converter (ADC).....</b>	<b>389</b>
<b>Overview.....</b>	<b>389</b>
<b>Block Diagram.....</b>	<b>389</b>
<b>Signal Description .....</b>	<b>390</b>
<b>Product Dependencies.....</b>	<b>390</b>
Power Management .....	390
Interrupt Sources.....	390
Analog Inputs .....	390
I/O Lines.....	390
Timer Triggers.....	390
Conversion Performances.....	390
<b>Functional Description.....</b>	<b>391</b>
Analog-to-digital Conversion .....	391
Conversion Reference .....	391
Conversion Resolution .....	391
Conversion Results .....	391
Conversion Triggers.....	393
Sleep Mode and Conversion Sequencer .....	394
ADC Timings.....	394
<b>Analog-to-digital Converter (ADC) User Interface.....</b>	<b>395</b>
ADC Control Register.....	396
ADC Mode Register .....	397
ADC Channel Enable Register.....	399
ADC Channel Disable Register.....	399
ADC Channel Status Register.....	400
ADC Status Register.....	401
ADC Last Converted Data Register .....	402
ADC Interrupt Enable Register.....	403
ADC Interrupt Disable Register.....	404

---

<b>AT91SAM7S32 Electrical Characteristics .....</b>	<b>405</b>
<b>Absolute Maximum Ratings.....</b>	<b>405</b>
<b>DC Characteristics.....</b>	<b>406</b>
<b>Power Consumption.....</b>	<b>408</b>
Power Consumption Versus Modes .....	408
Peripheral Power Consumption in Active Mode.....	410
Power Consumption versus Master Clock Frequency in Active Mode.....	411
Power Consumption versus Master Clock Frequency in Ultra Low-power Mode .....	412
<b>Crystal Oscillators Characteristics.....</b>	<b>413</b>
RC Oscillator Characteristics .....	413
Main Oscillator Characteristics.....	413

XIN Clock Characteristics .....	413
<b>PLL Characteristics .....</b>	<b>414</b>
<b>ADC Characteristics .....</b>	<b>415</b>
<hr/>	
<b>AT91SAM7S32 AC Characteristics .....</b>	<b>416</b>
<b>Applicable Conditions and Derating Data .....</b>	<b>416</b>
Conditions and Timings Computation .....	416
Temperature Derating Factor .....	417
VDDCORE Voltage Derating Factor .....	417
VDDIO Voltage Derating Factor .....	418
<b>Clock Characteristics .....</b>	<b>419</b>
Master Clock Characteristics .....	419
<b>Embedded Flash Characteristics .....</b>	<b>420</b>
<b>JTAG/ICE Timings .....</b>	<b>421</b>
ICE Interface Signals .....	421
JTAG Interface Signals .....	422
<hr/>	
<b>AT91SAM7S32 Mechanical Characteristics .....</b>	<b>424</b>
<b>Thermal Considerations .....</b>	<b>424</b>
Thermal Data .....	424
Junction Temperature .....	425
<b>Package Drawings .....</b>	<b>426</b>
<b>Soldering Profile .....</b>	<b>428</b>
<b>AT91SAM7S32 Ordering Information .....</b>	<b>429</b>
<hr/>	
<b>Table of Contents i</b>	
<b>Revision History .....</b>	<b>xiv</b>





## Revision History

Doc. Rev.	Comments
6071A	•Date: 28-Oct-04



## 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 Arsenaux 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/Imaging/Hi-Rel MPU/ High Speed Converters/RF Datacom

Avenue de Rochepleine  
BP 123  
38521 Saint-Egreve Cedex, France  
Tel: (33) 4-76-58-30-00  
Fax: (33) 4-76-58-34-80



## 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. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.

© Atmel Corporation 2004. All rights reserved. ATMEL® and combinations thereof and DataFlash® are the registered trademarks of Atmel Corporation or its subsidiaries. Everywhere You Are<sup>SM</sup> is the trademark of Atmel Corporation or its subsidiaries.

ARM®, ARM Powered®, ARM7TDMI® and Thumb® are the registered trademarks and ARM9TDMI™, ARM920T™ and AMBA™ are the trademarks of ARM Ltd.; CompactFlash® is a registered trademark of the CompactFlash Association; SmartMedia™ is a trademark of the Solid State Floppy Disk Card Forum. Other terms and product names may be the trademarks of others.



Printed on recycled paper.

6071A-ATARM-28-Oct-04

