Features

- Incorporates the ARM7TDMI™ ARM® Thumb® Processor
  - Embedded ICE In-circuit Emulation, Debug Communication Channel Support
- 96K Bytes of Internal High-speed SRAM
- 256K Bytes of Internal High-speed ROM Integrating Default Boot Program
  - Downloads Application from External Storage Medium in Internal SRAM
- Memory Controller (MC)
  - Memory Protection Unit, Abort Status and Misalignment Detection
- Clock Generator and Power Management Controller (PMC)
  - 3 to 20 MHz and 32 kHz On-chip Oscillators with Two PLLs
  - Programmable Software Power Optimization Capabilities
  - Four Programmable External Clock Signals
- Advanced Interrupt Controller (AIC)
  - Thirty Individually Maskable, Eight-level Priority, Vectored Interrupt Sources
  - Seven External Interrupt Sources and One Fast Interrupt Source, Spurious Interrupt Protected
- Two 32-bit Parallel Input/Output Controllers (PIO) PIOA and PIOB
  - Sixty-three 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 and Synchronous Output
- System Timer (ST) Including a 16-bit Counter, Watchdog and Second Counter
- Real Time Clock (RTC) with Alarm Interrupt
- Debug Unit (DBGU), 2-wire USART and Support for Debug Communication Channel
  - Programmable ICE Access Prevention
- Twenty Peripheral Data Controller (PDC) Channels
- USB 2.0 Full-speed (12 Mbits per second) Device Port (UDP)
  - On-chip Transceiver
  - 2-Kbyte Configurable FIFO for Loading and Storing Messages
- Multimedia Card Interface (MCI)
  - Automatic Protocol Control and Fast Automatic Data Transfers with PDC
  - MMC and SDCard Compliant, Support for up to two SDCards
- Three Synchronous Serial Controllers (SSC)
  - Independent Clock and Frame Sync Signals for Each Receiver and Transmitter
  - I²S Analog Interface Support, Time Division Multiplex Support
  - High-speed Continuous Data Stream Capabilities with 32-bit Data Transfer
- Four Universal Synchronous/Asynchronous Receiver Transmitters (USART)
  - Individual Baud Rate Generator
  - Support for ISO7816 T0/T1 Smart Card, Hardware and Software Handshaking, RS485 Support
  - Modem Control Lines on USART 1, IrDA Infrared Modulation/Demodulation
- Master/Slave Serial Peripheral Interface (SPI)
  - 8- to 16-bit Programmable Data Length
  - Four External Peripheral Chip Selects
- Two Three-channel 16-bit Timer/Counters (TC)
  - Three External Clock Inputs, Two Multi-purpose I/O Pins per Channel
  - Double PWM Generation, Capture/Waveform Mode, Up/Down Capability
- Two-wire Interface (TWI)
  - Master Mode Support, All Two-wire Atmel EEPROMs Supported
- IEEE 1149.1 JTAG Boundary Scan on All Digital Pins
- Required Power Supplies:
  - 1.65V to 1.95V for VDDCORE, VDDOSC and VDDPLL
  - 1.65V to 3.6V on VDDIO
- Fully Static Operation: 0 Hz to 66 MHz @2.7V/1.8V, up to 60 MIPS
- Available in a 100-lead LQFP Package

ARM7TDMI™-based Microcontroller

AT91RM3400

1790A–ATARM–11/03

Description

The AT91RM3400 is a fully integrated member of the Atmel advanced AT91 ARM microcontroller family. Having no external memory interface and equipped with embedded SRAM and ROM, it is ideal for numerous applications with medium memory requirements but which demand high performance.

Several options are available to download software to the internal SRAM. These include downloading from a serial EEPROM or serial DataFlash® or downloading through the USB Device Port. Additionally, customizing of the embedded ROM is available on request for large volume opportunities.

The Advanced Interrupt Controller (AIC) enhances the interrupt handling performance of the ARM7TDMI processor by providing multiple vectored, prioritized interrupt sources and reduces the cycles taken to transfer to an interrupt handler.

The Peripheral Data Controller (PDC) provides DMA channels for all the serial peripherals, enabling them to transfer data to or from on-chip memories without processor intervention. This reduces the processor overhead when dealing with transfers of continuous data streams.

The set of Parallel I/O (PIO) Controllers multiplex the peripheral input/output lines with general-purpose data I/Os, reducing the external pin count of the device and providing an interrupt and open drain capability on each line.

The Power Management Controller (PMC) keeps system power consumption to a minimum by selectively enabling and/or disabling the core and various peripherals under software control. It uses an enhanced clock generator to provide a selection of clock signals including a slow clock (32 kHz) for power-saving mode.

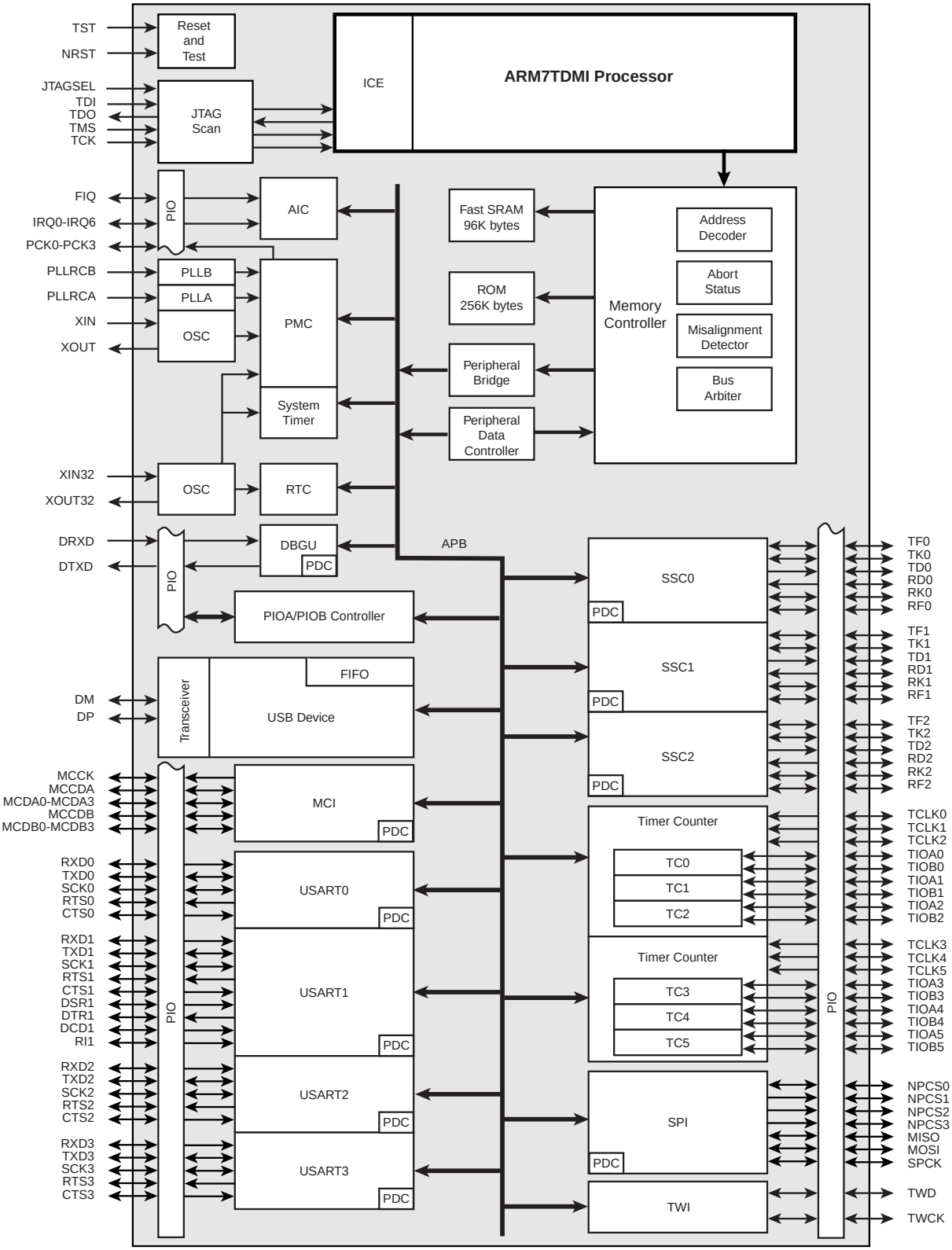
The wide range of system interfaces includes USB V2.0 Full-speed Device Port, Multimedia Card, Serial Peripheral Interface (SPI) and Two-wire Interface (TWI). Peripherals include multiple USARTs, Timer/Counters and Serial Synchronous Controllers (SSC).

The AT91RM3400 includes an extensive set of peripherals that operate in accordance with several industry standards, such as those used in audio, communication, computer and smart card applications.

Block Diagram

Bold arrows (⟶) indicate master-to-slave dependency.

Figure 1. AT91RM3400 Block Diagram



Key Features

ARM7TDMI Processor

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

Debug and Test

- Integrated Embedded In-circuit Emulator
- Debug Unit
  - Two-pin UART
  - Debug Communication Channel
  - Chip ID Register
- IEEE1149.1 JTAG Boundary-scan on All Digital Pins

Boot ROM Program

- Default Boot Program stored in ROM-based products
- Downloads and runs an application from external storage media into internal SRAM
- Downloaded code size depends on embedded SRAM size
- Automatic detection of valid application
- Bootloader supporting a wide range of non-volatile memories
  - SPI DataFlash® connected on SPI NPCS0
  - Two-wire EEPROM
- Boot Uploader in case no valid program is detected in external NVM and supporting several communication media
- Serial communication on a DBGU (XModem protocol)
- USB Device Port (DFU Protocol)

Embedded Software Services

- Compliant with ATPCS
- Compliant with ANSI/ISO Standard C
- Compiled in ARM/Thumb Interworking
- ROM Entry Service
- Tempo, Xmodem and DataFlash services
- CRC and Sine tables

Reset Controller

- One reset line providing
  - Initialization of the User Interface registers (defined in the user interface of each peripheral) and sampling of the signals needed at bootup. It forces the processor to fetch the next instruction at address zero.
  - Initialization of the embedded ICE TAP controller.

Memory Controller

- Bus Arbiter

  - Handles Requests from the ARM7TDMI and the Peripheral Data Controller
- Address Decoder Provides Selection Signals for
  - Up to Four 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
  - Allows Remapping of an Internal SRAM in Place of the Internal ROM
  - Allows Handling of Dynamic Interrupt Vectors
- 16-area Memory Protection Unit
  - Individually Programmable Size Between 1K Bytes and 64M Bytes
  - Individually Programmable Protection Against Write and/or User Access
  - Peripheral Protection Against Write and/or User Access

Advanced Interrupt Controller

- Controls the Interrupt Lines (nIRQ and nFIQ) of an ARM® Processor
- Thirty-two Individually Maskable and Vectored Interrupt Sources
  - Source 0 is Reserved for the Fast Interrupt Input (FIQ)
  - Source 1 is Reserved for System Peripherals (e.g., ST, RTC, PMC, DBGU)
  - Source 2 to Source 31 Control up to Thirty Embedded 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 1 to 31
  - 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 when Protect ModeIs Are Enabled
- Fast Forcing
  - Permits Redirecting any Normal Interrupt Source on the Fast Interrupt of the Processor
- General Interrupt Mask

- Provides Processor Synchronization on Events Without Triggering an Interrupt

Power Management Controller

- Optimizes the Power Consumption of the Whole System
- Embeds and Controls
  - One Main Oscillator and One Slow Clock Oscillator (32.768Hz)
  - Two Phase Locked Loops (PLLs) and Dividers
  - Clock Prescalers
- Provides
  - the Processor Clock PCK
  - the Master Clock MCK
  - Up to two USB Clocks (depending on the USB ports embedded)
    - UHPCK for the USB Host Port
    - UDPCK for the USB Device Port
  - Programmable Automatic PLL Switch-off in USB Device Suspend Conditions
  - Up to Thirty Peripheral Clocks
  - Up to Four Programmable Clock Outputs
- Four Operating Modes
  - Normal Mode, Idle Mode, Slow Clock Mode, Standby Mode

System Timer

- One Period Interval Timer, 16-bit Programmable Counter
- One Watchdog Timer, 16-bit Programmable Counter
- One Real-time Timer, 20-bit Free-running Counter
- Interrupt Generation on Event

Real-time Clock

- Low Power Consumption
- Full Asynchronous Design
- Two Hundred Year Calendar
- Programmable Periodic Interrupt
- Alarm and Update Parallel Load
- Control of Alarm and Update Time/Calendar Data In

Debug Unit

- System Peripheral to Facilitate Debug of Atmel's ARM-based Systems
- Composed of Four Functions
  - Two-pin UART
  - Debug Communication Channel (DCC) Support
  - Chip ID Registers
  - ICE Access Prevention
- Two-pin UART
  - Implemented Features are 100% Compatible with the Standard Atmel USART
  - Independent Receiver and Transmitter with a Common Programmable Baud Rate Generator
  - Even, Odd, Mark or Space Parity Generation

  - Parity, Framing and Overrun Error Detection
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
  - Interrupt Generation
  - Support for Two PDC Channels with Connection to Receiver and Transmitter
- Debug Communication Channel Support
  - Offers Visibility of COMMRX and COMMTX Signals from the ARM Processor
  - Interrupt Generation
- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals
- ICE Access Prevention
  - Enables Software to Prevent System Access Through the ARM Processor's ICE
  - Prevention is Made by Asserting the NTRST Line of the ARM Processor's ICE

Parallel Input/Output Controller

- Up to 32 Programmable I/O Lines
- Fully Programmable through Set/Clear Registers
- Multiplexing of Two Peripheral Functions per I/O Line
- For each I/O Line (Whether Assigned to a Peripheral or Used as General Purpose I/O)
  - Input Change Interrupt
  - 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

Serial Peripheral Interface

- Supports Communication with Serial External Devices
  - 4 Chip Selects with External Decoder Support Allow Communication with Up to 15 Peripherals
  - Serial Memories, such as DataFlash and 3-wire EEPROMs
  - Serial Peripherals, such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - External Co-processors
- Master or Slave Serial Peripheral Bus Interface
  - 8- to 16-bit Programmable Data Length Per Chip Select
  - Programmable Phase and Polarity Per Chip Select
  - Programmable Transfer Delays Between Consecutive Transfers and Between Clock and Data Per Chip Select
  - Programmable Delay Between Consecutive Transfers
  - Selectable Mode Fault Detection
- Connection to PDC Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
  - Next Buffer Support

Two-wire Interface

- Compatibility with standard two-wire serial memory
- 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 or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by-16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Optional Modem Signal Management DTR-DSR-DCD-RI
  - Receiver Time-out and Transmitter Timeguard
  - Optional Multi-Drop Mode with Address Generation and Detection
- RS485 with driver control signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation
  - Communication at up to 115.2 Kbps
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of Two Peripheral Data Controller Channels (PDC)
  - Offers Buffer Transfer without Processor Intervention

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
- Interfaced with Two PDC Channels (DMA Access) to Reduce Processor Overhead
- 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
- A 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
  - Two Multi-purpose Input/Output Signals
- Internal Interrupt Signal
- Two Global Registers that Act on All Three TC Channels

Multimedia Card Interface

- Compatibility with MultiMedia Card Specification Version 2.2
- Compatibility with SD Memory Card Specification Version 1.0
- Cards clock rate up to Master Clock divided by 2
- Embedded power management to slow down clock rate when not used
- Supports up to sixteen multiplexed slots (product-dependent)
  - One slot for one MultiMediaCard bus (up to 30 cards) or one SD Memory Card
- Support for stream, block and multi-block data read and write
- Supports connection to Peripheral Data Controller
  - Minimizes processor intervention for large buffer transfers

USB Device Port

- USB V2.0 Full-speed Compliant, 12 Mbits per second
- Embedded USB V2.0 Full-speed Transceiver
- Embedded Dual-port RAM for Endpoints
- Suspend/Resume Logic
- Ping-pong Mode (2 Memory Banks) for Isochronous and Bulk Endpoints

AT91RM3400 Product Properties

Power Supplies

The AT91RM3400 has four types of power supply pins:

- VDDCORE pins power the chip core and must be between 1.65V and 1.95V, 1.8V nominal.
- VDDIO pins power the I/O lines and must be between 1.65V and 3.6V, 1.8V, 3V or 3.3V nominal.
- VDDPLL pin powers the PLL cells and must be between 1.65V and 1.95V, 1.8V nominal.
- VDDOSC pin powers both oscillators and must be between 1.65V and 1.95V, 1.8V nominal.

Ground pins are common for all power supplies except the VDDPLL and VDDOSC, for which the GNDPLL and the GNDOSC pins are provided, respectively.

Powering VDDIO with a voltage lower than 3V prevents any use of the USB Device Port.

Pinout

The AT91RM3400 is available in a 100-lead LQFP package.

Table 1. AT91RM3400 Pinout in 100-lead LQFP Package

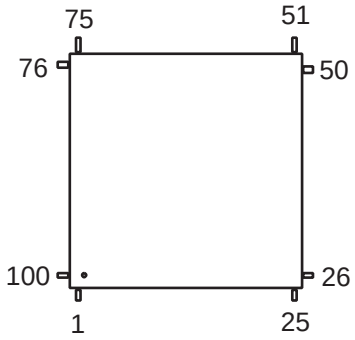
1	VDDCORE	26	PA11	51	PA30	76	PB21
2	GND	27	PA12	52	PA31	77	PB22
3	VDDPLL	28	PA13	53	PB0	78	JTAGSEL
4	PLLRCB	29	VDDIO	54	PB1	79	TDI
5	GNDPLL	30	GND	55	PB2	80	TDO
6	XOUT	31	PA14	56	PB3	81	TCK
7	XIN	32	PA15	57	PB4	82	TMS
8	VDDOSC	33	PA16	58	PB5	83	VDDIO
9	GNDOSC	34	PA17	59	PB6	84	GND
10	XOUT32	35	VDDCORE	60	PB7	85	TST
11	XIN32	36	GND	61	PB8	86	NRST
12	VDDPLL	37	PA18	62	PB9	87	VDDCORE
13	PLLRCA	38	PA19	63	PB10	88	GND
14	GNDPLL	39	PA20	64	PB11	89	PB23
15	PA0	40	PA21	65	PB12	90	PB24
16	PA1	41	PA22	66	VDDIO	91	PB25
17	PA2	42	PA23	67	GND	92	PB26
18	PA3	43	PA24	68	PB13	93	PB27
19	PA4	44	PA25	69	PB14	94	PB28
20	PA5	45	PA26	70	PB15	95	PB29
21	PA6	46	PA27	71	PB16	96	PB30
22	PA7	47	PA28	72	PB17	97	DDM
23	PA8	48	VDDIO	73	PB18	98	DDP
24	PA9	49	GND	74	PB19	99	VDDIO
25	PA10	50	PA29	75	PB20	100	GND

Mechanical Overview of the 100-lead LQFP Package

Figure 2 shows the orientation of the 100-lead LQFP package.

A detailed mechanical description is given in the section Mechanical Characteristics of the product datasheet.

Figure 2. 100-lead LQFP Pinout (Top View)



Peripheral Multiplexing on PIO Lines

The AT91RM3400 features two PIO controllers (PIOA and PIOB) that allow multiplexing of the I/O lines of the peripheral set.

Each PIO controller controls up to 32 lines. Each line can be assigned to one of the two peripheral functions, A or B.

The tables in the following paragraphs define how the I/O lines of the peripheral A and B are multiplexed on the PIO controllers A and B. The two columns “Function” and “Comments” have been inserted for the user’s own comments; they may be used to track how pins are defined in an application.

PIO Controller A Multiplexing

Table 2. Multiplexing on PIO Controller A

PIO Controller A			Application Usage	
I/O Line	**Peripheral A**	**Peripheral B**	**Function**	**Comments**
PA0	MISO	–		
PA1	MOSI	–		
PA2	SPCK	PCK0		
PA3	NPCS0	PCK1		
PA4	NPCS1	–		
PA5	NPCS2	SCK1		
PA6	NPCS3	SCK2		
PA7	TWD	PCK2		
PA8	TWCK	PCK3		
PA9	TXD0	–		
PA10	RXD0	–		
PA11	SCK0	TCLK0		
PA12	CTS0	TCLK1		
PA13	RTS0	TCLK2		
PA14	RXD1	–		
PA15	TXD1	–		
PA16	RTS1	TIOA0		
PA17	CTS1	TIOB0		
PA18	DTR1	TIOA1		
PA19	DSR1	TIOB1		
PA20	DCD1	TIOA2		
PA21	RI1	TIOB2		
PA22	RXD2	–		
PA23	TXD2	–		
PA24	MCCK	RTS0		
PA25	MCCDA	RTS1		

Table 2. Multiplexing on PIO Controller A (Continued)

PIO Controller A			Application Usage	
I/O Line	**Peripheral A**	**Peripheral B**	**Function**	**Comments**
PA26	MCDA0			
PA27	MCDA1	–		
PA28	MCDA2	RTS2		
PA29	MCDA3	CTS2		
PA30	DRXD	–		
PA31	DTXD	–		

PIO Controller B Multiplexing

Table 3. Multiplexing PIO controller B

PIO Controller B			Application Usage	
I/O Line	Peripheral A	Peripheral B	Function	Comments
PB0	TF0	TIOB3		
PB1	TK0	TCLK3		
PB2	TD0	RTS2		
PB3	RD0	RTS3		
PB4	RK0	PCK0		
PB5	RF0	TIOA3		
PB6	TF1	TIOB4		
PB7	TK1	TCLK4		
PB8	TD1	NPCS1		
PB9	RD1	NPCS2		
PB10	RK1	PCK1		
PB11	RF1	TIOA4		
PB12	TF2	TIOB5		
PB13	TK2	TCLK5		
PB14	TD2	NPCS3		
PB15	RD2	PCK1		
PB16	RK2	PCK2		
PB17	RF2	TIOA5		
PB18	RTS3	MCCDB		
PB19	CTS3	MCDB0		
PB20	TXD3	DTR1		
PB21	RXD3			
PB22	SCK3	PCK3		
PB23	FIQ			
PB24	IRQ0	TD0		
PB25	IRQ1	TD1		
PB26	IRQ2	TD2		
PB27	IRQ3	DTXD		
PB28	IRQ4	MCDB1		
PB29	IRQ5	MCDB2		
PB30	IRQ6	MCDB3		

Pin Name Description

Table 4 gives details on the pin name classified by peripheral.

Table 4. Pin Description List

Pin Name	Function	Type	Active Level	Comments
Power				
VDDIO	Memory I/O Lines Power Supply	Power		1.65V to 3.6V
VDDPLL	Oscillator and PLL Power Supply	Power		1.65V to 1.95V
VDDCORE	Core Chip Power Supply	Power		1.65V to 1.95V
VDDOSC	Oscillator Power Supply	Power		1.65V to 1.95V
GND	Ground	Ground		
GNDPLL	PLL Ground	Ground		
GNDOSC	Oscillator Ground	Ground		
Clock Generation and Power Management (PMC)				
XIN	Main Crystal Input	Input		
XOUT	Main Crystal Output	Output		
XIN32	32KHz Crystal Input	Input		
XOUT32	32KHz Crystal Output	Output		
PLLRCA	PLL A Filter	Input		
PLLRCB	PLL B Filter	Input		
PCK0 - PCK3	Programmable Clock Output	Output		
ICE and JTAG				
TCK	Test Clock	Input		
TDI	Test Data In	Input		
TDO	Test Data Out	Output		
TMS	Test Mode Select	Input		
JTAGSEL	JTAG Selection	Input		
Reset/Test				
NRST	Microcontroller Reset	Input	Low	No on-chip pull-up
TST	Test Mode Select	Input		Must be tied low for normal operation
Debug Unit				
DRXD	Debug Receive Data	Input		
DTXD	Debug Transmit Data	Output		
AIC				
IRQ0 - IRQ6	Interrupt Inputs	Input		
FIQ	Fast Interrupt Input	Input		

Table 4. Pin Description List

Pin Name	Function	Type	Active Level	Comments
PIO				
PA0 - PA31	Parallel IO Controller A	I/O		Pulled-up input at reset
PB0 - PB30	Parallel IO Controller B	I/O		Pulled-up input at reset
Multi-media Card Interface				
MCCK	Multimedia Card Clock	Output		
MCCDA	Multimedia Card A Command	I/O		
MCDA0 - MCDA3	Multimedia Card A Data	I/O		
MCCDB	Multimedia Card B Command	I/O		
MCDB0 - MCDB3	Multimedia Card B Data	I/O		
USART				
SCK0 - SCK3	Serial Clock	I/O		
TXD0 - TXD3	Transmit Data	I/O		
RXD0 - RXD3	Receive Data	Input		
RTS0 - RTS3	Ready To Send	Output		
CTS0 - CTS3	Clear To Send	Input		
DSR1	Data Set Ready	Input		
DTR1	Data Terminal Ready	Output		
DCD1	Data Carrier Detect	Input		
RI1	Ring Indicator	Input		
USB Device Port				
DM	USB Device Port Data -	Analog		
DP	USB Device Port Data +	Analog		
Synchronous Serial Controller				
TD0 - TD2	Transmit Data	Output		
RD0 - RD2	Receive Data	Input		
TK0 - TK2	Transmit Clock	I/O		
RK0 - RK2	Receive Clock	I/O		
TF0 - TF2	Transmit Frame Sync	I/O		
RF0 - RF2	Receive Frame Sync	I/O		
Timer/Counter				
TCLK0 - TCLK5	External Clock Input	Input		
TIOA0 - TIOA5	Multipurpose Timer I/O Pin A	I/O		
TIOB0 - TIOB5	Multipurpose Timer I/O Pin B	I/O		

Table 4. Pin Description List

Pin Name	Function	Type	Active Level	Comments
SPI				
MISO	Master In Slave Out	I/O		
MOSI	Master Out Slave In	I/O		
SPCK	SPI Serial Clock	I/O		
NPCS0 - NPCS3	SPI Peripheral Chip Select 0 to 3	I/O	Low	
Two-wire Interface				
TWD	Two-wire Serial Data	I/O		
TWCK	Two-wire Serial Clock	I/O		

Peripheral Identifiers

The AT91RM3400 embeds a wide range of peripherals. Table 5 defines the Peripherals Identifiers of the AT91RM3400. 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 5. Peripheral Identifiers

Peripheral ID	Peripheral Mnemonic	Peripheral Name	External Interrupt
0	AIC	Advanced Interrupt Controller	FIQ
1	SYSIRQ	System Interrupt	
2	PIOA	Parallel IO Controller A	
3	PIOB	Parallel IO Controller B	
4	–	Reserved	
5	–	Reserved	
6	US0	USART 0	
7	US1	USART 1	
8	US2	USART 2	
9	US3	USART 3	
10	MCI	Multimedia Card Interface	
11	UDP	USB Device Port	
12	TWI	Two-Wire Interface	
13	SPI	Serial Peripheral Interface	
14	SSC0	Serial Synchronous Controller 0	
15	SSC1	Serial Synchronous Controller 1	
16	SSC2	Serial Synchronous Controller 2	
17	TC0	Timer Counter 0	
18	TC1	Timer Counter 1	
19	TC2	Timer Counter 2	
20	TC3	Timer Counter 3	
21	TC4	Timer Counter 4	
22	TC5	Timer Counter 5	
23	–	Reserved	
24	–	Reserved	
25	AIC	Advanced Interrupt Controller	IRQ0
26	AIC	Advanced Interrupt Controller	IRQ1
27	AIC	Advanced Interrupt Controller	IRQ2
28	AIC	Advanced Interrupt Controller	IRQ3
29	AIC	Advanced Interrupt Controller	IRQ4
30	AIC	Advanced Interrupt Controller	IRQ5
31	AIC	Advanced Interrupt Controller	IRQ6

System Interrupt

The system interrupt (Peripheral ID 1) is the wired-OR of the signal coming from:

- the Power Management Controller
- the System Timer
- the Real Time Clock
- the Debug Unit

The clock of these peripherals cannot be controlled and the Peripheral ID 1 can only be used within the Advanced Interrupt Controller.

External Interrupts

All external interrupt signals, i.e, the Fast Interrupt signal FIQ or the Interrupt signals IRQ0 to IRQ6, use a dedicated Peripheral ID. However, there is no clock control associated with these peripherals IDs.

Product Memory Mapping

Internal Memory Mapping

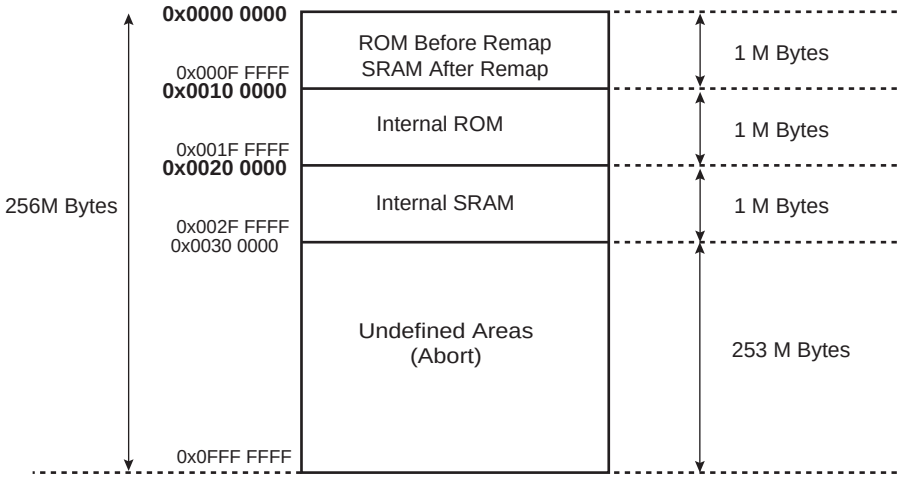
Internal RAM

The AT91RM3400 embeds a high-speed 96-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 ROM

The AT91RM3400 features one bank of 256K bytes of ROM. At any time, the ROM is mapped to address 0x0010 0000. It is also accessible at address 0x0 after the reset and before the Remap Command.

Figure 3. Internal Memory Mapping

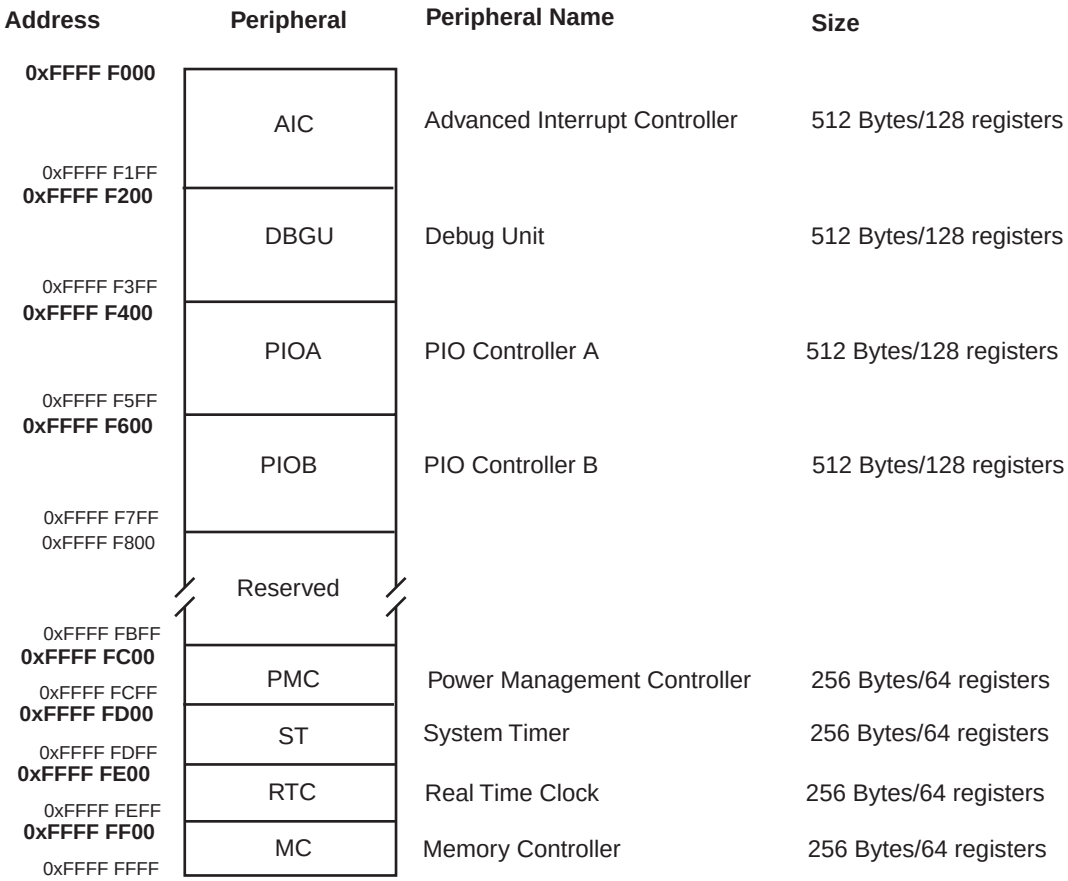


Peripheral Mapping

System Peripherals Mapping

The System Peripherals are all mapped to the highest 4K bytes of address space, between addresses 0xFFFF F000 and 0xFFFF FFFF. Each peripheral has an address space of 256 or 512 bytes, representing 64 or 128 registers.

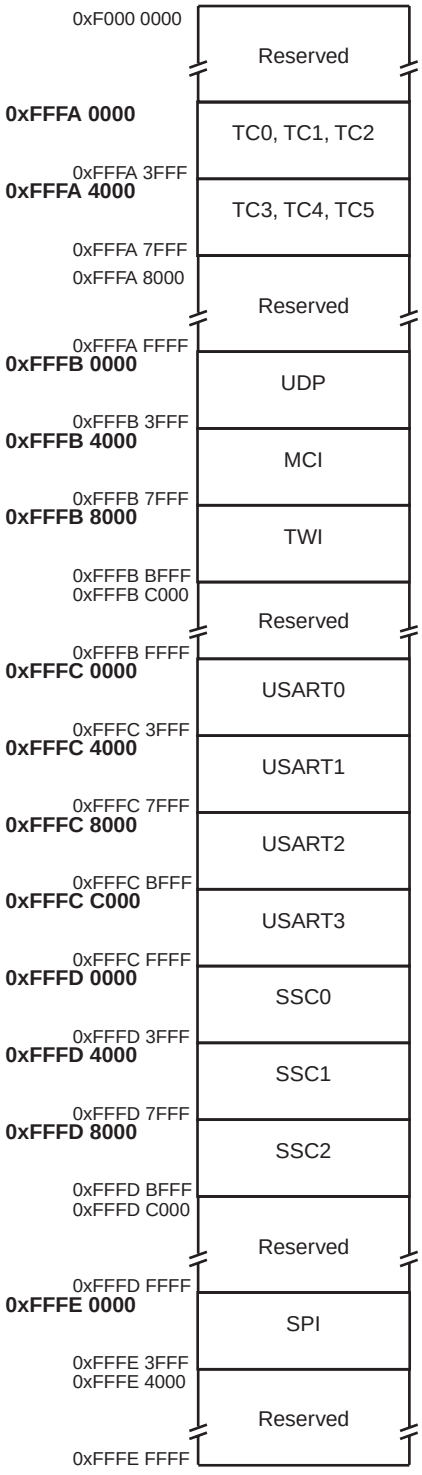
Figure 4. System Peripherals Mapping



User Peripherals Mapping

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

Figure 5. User Peripherals Mapping



Peripheral Name		Size
TC0, TC1, TC2	Timer/Counter 0, 1 and 2	16K Bytes
TC3, TC4, TC5	Timer/Counter 3, 4 and 5	16K Bytes
UDP	USB Device Port	16K Bytes
MCI	Multimedia Card Interface	16K Bytes
TWI	Two-Wire Interface	16K Bytes
USART0	Universal Synchronous Asynchronous Receiver Transmitter 0	16K Bytes
USART1	Universal Synchronous Asynchronous Receiver Transmitter 1	16K Bytes
USART2	Universal Synchronous Asynchronous Receiver Transmitter 2	16K Bytes
USART3	Universal Synchronous Asynchronous Receiver Transmitter 3	16K Bytes
SSC0	Serial Synchronous Controller 0	16K Bytes
SSC1	Serial Synchronous Controller 1	16K Bytes
SSC2	Serial Synchronous Controller 2	16K Bytes
SPI	Serial Peripheral Interface	16K Bytes

Peripheral Implementation

USART

The USART section describes features allowing management of the Modem Signals DTR, DSR, DCD and RI.

In the AT91RM3400, only the USART1 implements these signals, named DTR1, DSR1, DCD1 and RI1.

The USART0, USART2 and USART3 do not implement all the modem signals. Only RTS and CTS (RTS0 and CTS0, RTS2 and CTS2, RTS3 and CTS3, respectively) are implemented in these USARTs for other features.

Thus, programming the USART0, USART2 or the USART3 in Modem Mode may lead to unpredictable results. In these USARTs, the commands relating to the Modem Mode have no effect and the status bits relating the status of the modem signals are never activated.

Timer Counter

The Timer Counter 0 to 5 are described with five generic clock inputs, TIMER_CLOCK1 to TIMER_CLOCK5. In the AT91RM3400, these clock inputs are connected to the Master Clock (MCK), to the Slow Clock (SLCK) and to divisions of the Master Clock.

Table 6 gives the correspondence between the Timer Counter clock inputs and clocks in the AT91RM3400. Each Timer Counter 0 to 5 displays the same configuration.

Table 6. 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	SLCK

USB Device Port

The USB device port is V2.0 full-speed compliant. It features six general purpose endpoints configured as follows:

Endpoint 0: 8 bytes, no support of ping-pong mode

Endpoint 1: 64 bytes, supports ping-pong mode

Endpoint 2: 64 bytes, supports ping-pong mode

Endpoint 3 : 8 bytes, no support of ping-pong mode

Endpoint 4: 256 bytes, supports ping-pong mode

Endpoint 5 : 256 bytes, supports ping-pong mode

ARM7TDMI Processor Overview

Overview

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

The main features of the ARM7tDMI processor are:

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

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 37registers:

- 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 7. 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 8 gives the ARM instruction mnemonic list.

Table 8. ARM Instruction Mnemonic List

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

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

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 9 gives the Thumb instruction mnemonic list.

Table 9. Thumb Instruction Mnemonic List

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

AT91RM3400 Debug and Test Features

Overview

The AT91RM3400 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.

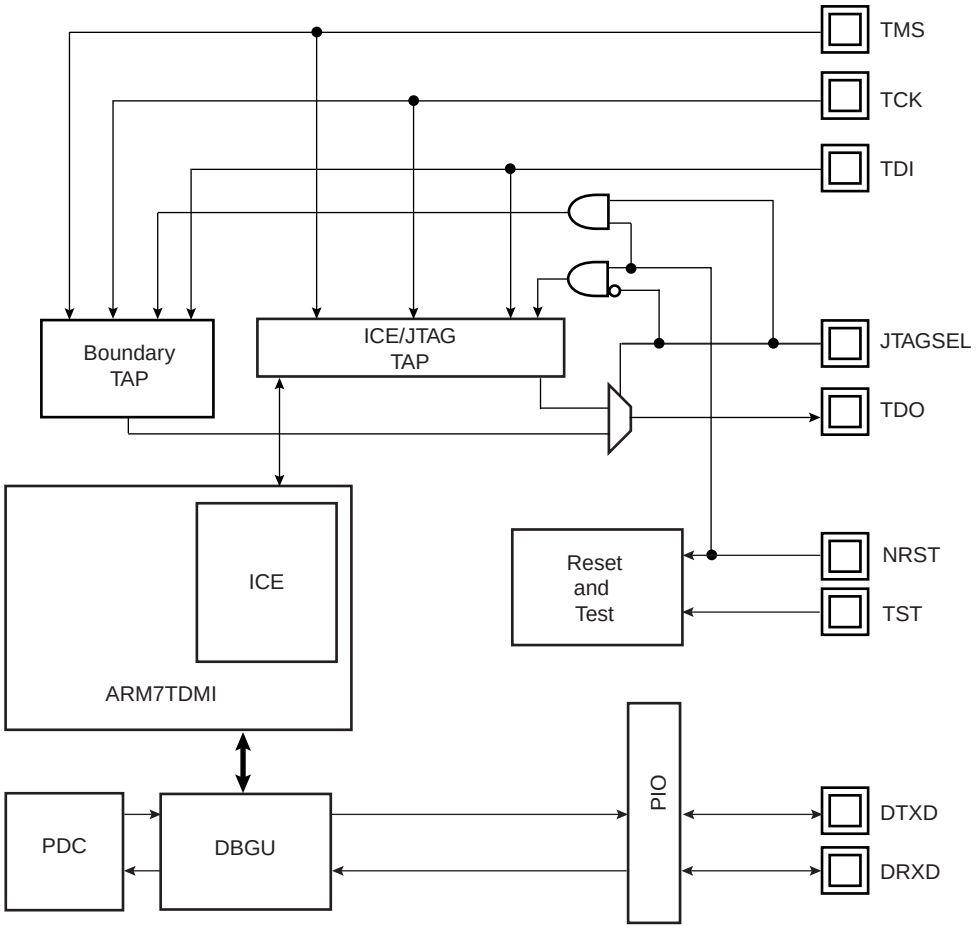
A set of dedicated debug and test input/output pins give direct access to these capabilities from a PC-based test environment.

Key features are:

- Integrated Embedded In-circuit Emulator
- Debug Unit
  - Two-pin UART
  - Debug Communication Channel
  - Chip ID Register
- IEEE1149.1 JTAG Boundary-scan on All Digital Pins

Block Diagram

Figure 6. AT91RM3400 Debug and Test Block Diagram



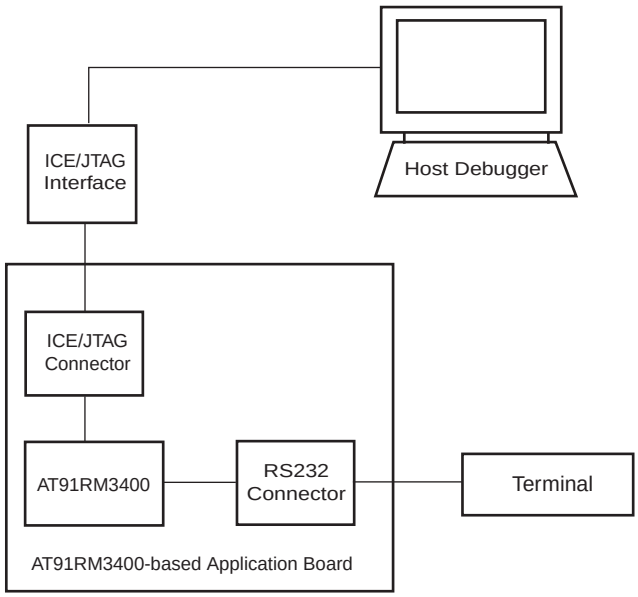
Note: TAP: Test Access Port

Application Examples

Debug Environment

Figure 7 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.

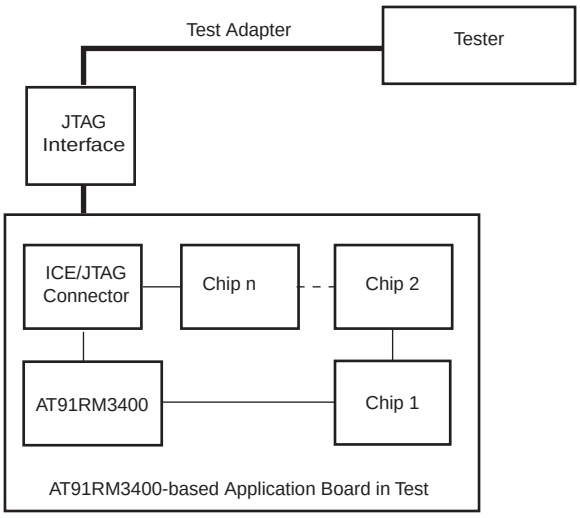
Figure 7. AT91RM3400-based Application Debug Environment Example



Test Environment

Figure 8 shows a test environment example. Test vectors are sent and interpreted by the tester. In this example, the “board under test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

Figure 8. AT91RM3400-based Application Test Environment Example



Debug and Test Pin Description

Table 10. Debug and Test Pin List

Pin Name	Function	Type	Active Level
Reset/Test			
NRST	Microcontroller Reset	Input	Low
TST	Test Mode Select	Input	
ICE and JTAG			
TCK	Test Clock	Input	
TDI	Test Data In	Input	
TDO	Test Data Out	Output	
TMS	Test Mode Select	Input	
JTAGSEL	JTAG Selection	Input	
Debug Unit			
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 to this pin are manufacturing test reserved.

Embedded In-circuit Emulator

ARM7TDMI embedded In-circuit Emulator is supported via the ICE/JTAG port.The internal state of the ARM7TDMI is examined through a 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 (NRST) 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, informs about the product version and its internal configuration.

AT91RM3400 Debug Unit Chip ID value is: 0x034E0941, on 32-bit width.

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

For further details on the Debug Unit and Boot program, see Boot Program Specifications.

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 (NRST) 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 189 bits which correspond to active pins and associated control signals.

Each AT91RM3400 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 11. JTAG Boundary Scan Register

Bit Number	Pin Name	Pin Type	Associated BSR Cells
189	PB18/RTS3/MCCDB	IN/OUT	INPUT
188			OUTPUT
187			CONTROL
186	PB19/CTS3/MCCDB0	IN/OUT	INPUT
185			OUTPUT
184			CONTROL

Table 11. JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
183	PB20/TXD3/DTR1	IN/OUT	INPUT
182			OUTPUT
181			CONTROL
180	PB21/RXD3	IN/OUT	INPUT
179			OUTPUT
178			CONTROL
177	PB22/SCK3/PCK3	IN/OUT	INPUT
176			OUTPUT
175			CONTROL
174	PB23/FIQ	IN/OUT	INPUT
173			OUTPUT
172			CONTROL
171	PB24/IRQ0/TD0	IN/OUT	INPUT
170			OUTPUT
169			CONTROL
168	PB25/IRQ1/TD1	IN/OUT	INPUT
167			OUTPUT
166			CONTROL
165	PB26/IRQ2/TD2	IN/OUT	INPUT
164			OUTPUT
163			CONTROL
162	PB27/IRQ3/DTXD	IN/OUT	INPUT
161			OUTPUT
160			CONTROL
159	PB28/IRQ4/MCDB1	IN/OUT	INPUT
158			OUTPUT
157			CONTROL
156	PB29/IRQ5/MCDB2	IN/OUT	INPUT
155			OUTPUT
154			CONTROL
153	PB30/IRQ6/MCDB3	IN/OUT	INPUT
152			OUTPUT
151			CONTROL

Table 11. JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
150	PA0/MISO	IN/OUT	INPUT
149			OUTPUT
148			CONTROL
147	PA1/MOSI	IN/OUT	INPUT
146			OUTPUT
145			CONTROL
144	PA2/SPCK/PCK0	IN/OUT	INPUT
143			OUTPUT
142			CONTROL
141	PA3/NPCS0/PCK1	IN/OUT	INPUT
140			OUTPUT
139			CONTROL
138	PA4/NPCS1	IN/OUT	INPUT
137			OUTPUT
136			CONTROL
135	PA5/NPCS2/SCK1	IN/OUT	INPUT
134			OUTPUT
133			CONTROL
132	PA6/NPCS3/SCK2	IN/OUT	INPUT
131			OUTPUT
130			CONTROL
129	PA7/TWD/PCK2	IN/OUT	INPUT
128			OUTPUT
127			CONTROL
126	PA8/TWCK/PCK3	IN/OUT	INPUT
125			OUTPUT
124			CONTROL
123	PA9/TXD0	IN/OUT	INPUT
122			OUTPUT
121			CONTROL
120	PA10/RXD0	IN/OUT	INPUT
119			OUTPUT
118			CONTROL

Table 11. JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
117	PA11/SCK0/TCLK0	IN/OUT	INPUT
116			OUTPUT
115			CONTROL
114	PA12/CTS0/TCLK1	IN/OUT	INPUT
113			OUTPUT
112			CONTROL
111	PA13/RTS0/TCLK2	IN/OUT	INPUT
110			OUTPUT
109			CONTROL
108	PA14/RXD1	IN/OUT	INPUT
107			OUTPUT
106			CONTROL
105	PA15/TXD1	IN/OUT	INPUT
104			OUTPUT
103			CONTROL
102	PA16/RTS1/TIOA0	IN/OUT	INPUT
101			OUTPUT
100			CONTROL
99	PA17/CTS1/TIOB0	IN/OUT	INPUT
98			OUTPUT
97			CONTROL
96	PA18/DTR1/TIOA1	IN/OUT	INPUT
95			OUTPUT
94			CONTROL
93	PA19/DSR1/TIOB1	IN/OUT	INPUT
92			OUTPUT
91			CONTROL
90	PA20/DCD1/TIOA2	IN/OUT	INPUT
89			OUTPUT
88			CONTROL
87	PA21/RI1/TIOB2	IN/OUT	INPUT
86			OUTPUT
85			CONTROL

Table 11. JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
84	PA22/RXD2	IN/OUT	INPUT
83			OUTPUT
82			CONTROL
81	PA23/TXD2	IN/OUT	INPUT
80			OUTPUT
79			CONTROL
78	PA24/MCCK/RTS0	IN/OUT	INPUT
77			OUTPUT
76			CONTROL
75	PA25/MCCDA/RTS1	IN/OUT	INPUT
74			OUTPUT
73			CONTROL
72	PA26/MCDA0	IN/OUT	INPUT
71			OUTPUT
70			CONTROL
69	PA27/MCDA1	IN/OUT	INPUT
68			OUTPUT
67			CONTROL
66	PA28/MCDA2/RTS2	IN/OUT	INPUT
65			OUTPUT
64			CONTROL
63	PA29/MCDA3/CTS2	IN/OUT	INPUT
62			OUTPUT
61			CONTROL
60	PA30/DRXD	IN/OUT	INPUT
59			OUTPUT
58			CONTROL
57	PA31/DTXD	IN/OUT	INPUT
56			OUTPUT
55			CONTROL
54	PB0/TF0/TIOB3	IN/OUT	INPUT
53			OUTPUT
52			CONTROL

Table 11. JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
51	PB1/TK0/TCLK3	IN/OUT	INPUT
50			OUTPUT
49			CONTROL
48	PB2/TD0/RTS2	IN/OUT	INPUT
47			OUTPUT
46			CONTROL
45	PB3/RD0/RTS3	IN/OUT	INPUT
44			OUTPUT
43			CONTROL
42	PB4/RK0/PCK0	IN/OUT	INPUT
41			OUTPUT
40			CONTROL
39	PB5/RF0/TIOA3	IN/OUT	INPUT
38			OUTPUT
37			CONTROL
36	PB6/TF1/TIOB4	IN/OUT	INPUT
35			OUTPUT
34			CONTROL
33	PB7/TK1/TCLK4	IN/OUT	INPUT
32			OUTPUT
31			CONTROL
30	PB8/TD1/NPCS1	IN/OUT	INPUT
29			OUTPUT
28			CONTROL
27	PB9/RD1/NPCS2	IN/OUT	INPUT
26			OUTPUT
25			CONTROL
24	PB10/RK1/PCK1	IN/OUT	INPUT
23			OUTPUT
22			CONTROL
21	PB11/RF1/TIOA4	IN/OUT	INPUT
20			OUTPUT
19			CONTROL

Table 11. JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
18	PB12/TF2/TIOB5	IN/OUT	INPUT
17			OUTPUT
16			CONTROL
15	PB13/TK2/TCLK5	IN/OUT	INPUT
14			OUTPUT
13			CONTROL
12	PB14/TD2/NPCS3	IN/OUT	INPUT
11			OUTPUT
10			CONTROL
9	PB15/RD2/PCK1	IN/OUT	INPUT
8			OUTPUT
7			CONTROL
6	PB16/RK2/PCK2	IN/OUT	INPUT
5			OUTPUT
4			CONTROL
3	PB17/RF2/TIOA5	IN/OUT	INPUT
2			OUTPUT
1			CONTROL

AT91RM3400 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: Product Version Number

Set to 0x1.

PART NUMBER: Product Part Number

Set to 0x5B03.

MANUFACTURER IDENTITY

Set to 0x01F.

Bit[0]

Required by IEEE Std. 1149.1.

Set to 0x1.

AT91RM3400 JTAG ID Code value is 0x15B0303F.

Boot Program

Overview

The Boot Program downloads an application in any of the AT91 products integrating a ROM. It integrates a Bootloader and a boot Uploader to assure correct information download.

The Bootloader is activated first. It looks for a sequence of eight valid ARM exception vectors in a DataFlash connected to the SPI, an EEPROM connected to the Two-wire Interface (TWI) or an 8-bit memory device connected to the external bus interface (EBI) (if the device integrates the EBI). All these vectors must be B-branch or LDR load register instructions except for the sixth instruction. This vector is used to store information, such as the size of the image to download and the type of DataFlash device.

If a valid sequence is found, code is downloaded into the internal SRAM. This is followed by a remap and a jump to the first address of the SRAM.

If no valid ARM vector sequence is found, the boot Uploader is started. It initializes the Debug Unit serial port (DBGU) and the USB Device Port. It then waits for any transaction and downloads a piece of code into the internal SRAM via a Device Firmware Upgrade (DFU) protocol for USB and XMODEM protocol for the DBGU. After the end of the download, it branches to the application entry point at the first address of the SRAM.

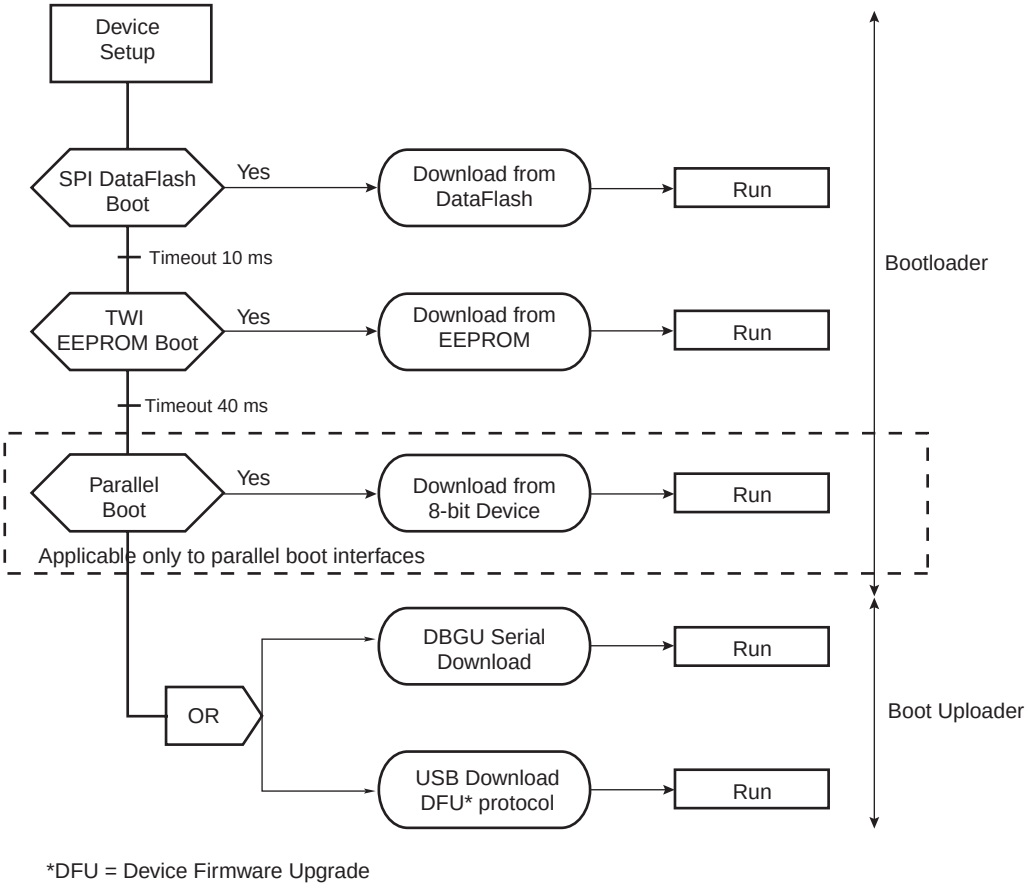
The main features of the Boot Program are:

- Default Boot Program stored in ROM-based products
- Downloads and runs an application from external storage media into internal SRAM
- Downloaded code size depends on embedded SRAM size
- Automatic detection of valid application
- Bootloader supporting a wide range of non-volatile memories
  - SPI DataFlash® connected on SPI NPCS0
  - Two-wire EEPROM
  - 8-bit parallel memories on NCS0 (only for devices with EBI integrated)
- Boot Uploader in case no valid program is detected in external NVM and supporting several communication media
- Serial communication on a DBGU (XModem protocol)
- USB Device Port (DFU Protocol)

Flow Diagram

The Boot Program implements the algorithm presented in Figure 9.

Figure 9. Boot Program Algorithm Flow Diagram



*DFU = Device Firmware Upgrade

Bootloader

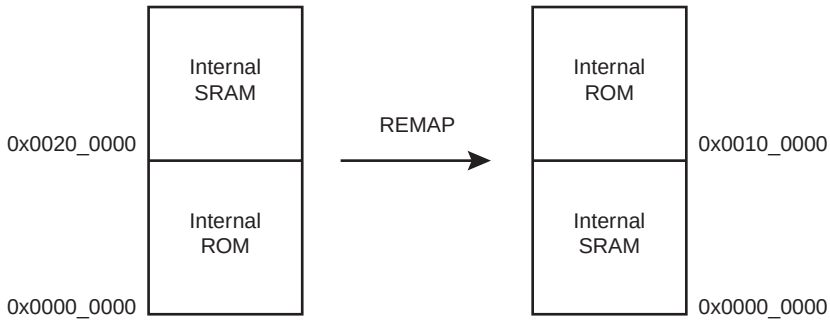
The Boot Program is started from address 0x0000_0000 (ARM reset vector) when the on-chip boot mode is selected (BMS high during the reset, only on devices with EBI integrated). The first operation is the search for a valid program in the off-chip non-volatile memories. If a valid application is found, this application is loaded into internal SRAM and executed by branching at address 0x0000_0000 after remap. This application may be the application code or a second-level Bootloader.

To optimize the downloaded application code size, the Boot Program embeds several functions that can be reused by the application. The Boot Program is linked at address 0x0010_0000 but the internal ROM is mapped at both 0x0000_0000 and 0x0010_0000 after reset. All the call to functions is PC relative and does not use absolute addresses. The ARM vectors are present at both addresses, 0x0000_0000 and 0x0010_0000.

To access the functions in ROM, a structure containing chip descriptor and function entry points is defined at a fixed address in ROM.

If no valid application is detected, the debug serial port or the USB device port must be connected to allow the upload. A specific application provided by Atmel (DFU uploader) loads the application into internal SRAM through the USB. To load the application through the debug serial port, a terminal application (HyperTerminal) running the Xmodem protocol is required.

Figure 10. Remap Action after Download Completion



After reset, the code in internal ROM is mapped at both addresses 0x0000_0000 and 0x0010_0000:

```
100000   ea00000b   B      0x2c              00   ea00000b   B      0x2c
100004   e59ff014   LDR    PC,[PC,20]        04   e59ff014   LDR    PC,[PC,20]
100008   e59ff014   LDR    PC,[PC,20]        08   e59ff014   LDR    PC,[PC,20]
10000c   e59ff014   LDR    PC,[PC,20]        0c   e59ff014   LDR    PC,[PC,20]
100010   e59ff014   LDR    PC,[PC,20]        10   e59ff014   LDR    PC,[PC,20]
100014   00001234   LDR    PC,[PC,20]        14   00001234   LDR    PC,[PC,20]
100018   e51fff20   LDR    PC,[PC,-0xf20]    18   e51fff20   LDR    PC,[PC,-0xf20]
10001c   e51fff20   LDR    PC,[PC,-0xf20]    1c   e51fff20   LDR    PC,[PC,-0xf20]
```

Valid Image Detection

The Bootloader software looks for a valid application by analyzing the first 32 bytes corresponding to the ARM exception vectors. These bytes must implement ARM instructions for either branch or load PC with PC relative addressing. The sixth vector, at offset 0x18, contains the size of the image to download and the DataFlash parameters.

The user must replace this vector with his own vector.

Figure 11. LDR Opcode

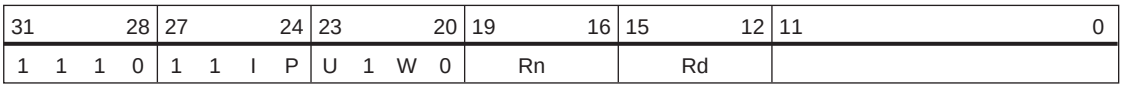
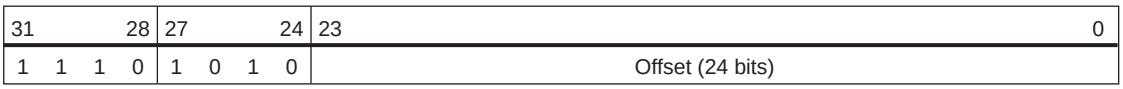


Figure 12. B Opcode



Unconditional instruction: 0xE for bits 31 to 28

Load PC with PC relative addressing instruction:

- Rn = Rd = PC = 0xF
- I==1
- P==1
- U offset added (U==1) or subtracted (U==0)
- W==1

Example

An example of valid vectors:

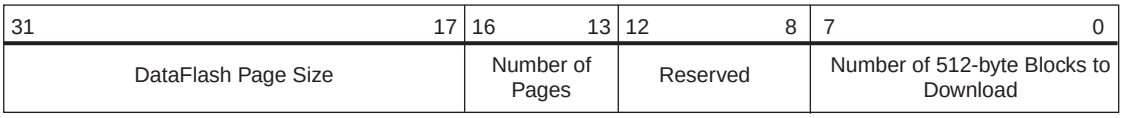
```
00      ea00000b        B           0x2c
004     e59ff014        LDR         PC, [PC,20]
08      e59ff014        LDR         PC, [PC,20]
0c      e59ff014        LDR         PC, [PC,20]
10      e59ff014        LDR         PC, [PC,20]
14      00001234        LDR         PC, [PC,20]     <- Code size = 4660 bytes
18      e51fff20        LDR         PC, [PC,-0xf20]
1c      e51fff20        LDR         PC, [PC,-0xf20]
```

In download mode (DataFlash, EEPROM or 8-bit memory in device with EBI integrated), the size of the image to load into SRAM is contained in the location of the sixth ARM vector. Thus the user must replace this vector by the correct vector for his application.

Structure of ARM Vector 6

The ARM exception vector 6 is used to store information needed by the Boot ROM downloader. This information is described below.

Figure 13. Structure of the ARM Vector 6



The first eight bits contain the number of blocks to download. The size of a block is 512 bytes, allowing download of up to 128K bytes.

The bits 13 to 16 determine the DataFlash page number.

– DataFlash page number = $2^{(\text{Nb of pages})}$

The last 15 bits contain the DataFlash page size.

Table 12. DataFlash Device

Device	Density	Page Size (bytes)	Number of Pages
AT45DB011B	1 Mbit	264	512
AT45DB021B	2 Mbits	264	1024
AT45DB041B	4 Mbits	264	2048
AT45DB081B	8 Mbits	264	4096
AT45DB161B	16 Mbits	528	4096
AT45DB321B	32 Mbits	528	8192
AT45DB642	64 Mbits	1056	8192
AT45DB1282	128 Mbits	1056	16384

Example

The following vector contains the information to describe a AT45DB642 DataFlash which contains 11776 bytes to download.

Vector 6 is 0x0841A017 (00001000010000011010000000010111b):

Size to download: 0x17 * 512 bytes = 11776 bytes

Number pages (1101b): 13 ==> Number of DataFlash pages = 2^{13} = 8192

DataFlash page size(000010000100000b) = 1056

For download in the EEPROM or 8-bit external memory (if device integrates EBI), only the size to be downloaded is decoded.

Bootloader Sequence

The Boot Program performs device initialization followed by the download procedure. If unsuccessful, the upload is done via the USB or debug serial port.

Device Initialization

Initialization follows the steps described below:

1. PLL setup
   – PLLB is initialized to generate a 48 MHz clock necessary to use the USB Device. A register located in the Power Management Controller (PMC) determines the frequency of the main oscillator and thus the correct factor for the PLLB. Table 13 defines the crystals supported by the Boot Program.

Table 13. Crystals Supported by Software Auto-detection (MHz)

3.0	3.2768	3.6864	3.84	4.0
4.433619	4.9152	5.0	5.24288	6.0
6.144	6.4	6.5536	7.159090	7.3728
7.864320	8.0	9.8304	10.0	11.05920
12.0	12.288	13.56	14.31818	14.7456
16.0	17.734470	18.432	20.0	24.0
25.0	28.224	32.0	33.0	

2. Stacks setup for each ARM mode
3. Main oscillator frequency detection
4. Interrupt controller setup
5. C variables initialization
6. Branch main function

Download Procedure

The download procedure checks for a valid boot on several devices. The first device checked is a serial DataFlash connected to the NPCS0 of the SPI, followed by the serial EEPROM connected to the TWI and by an 8-bit parallel memory connected on NCS0 of the External Bus Interface (if EBI is implemented in the product).

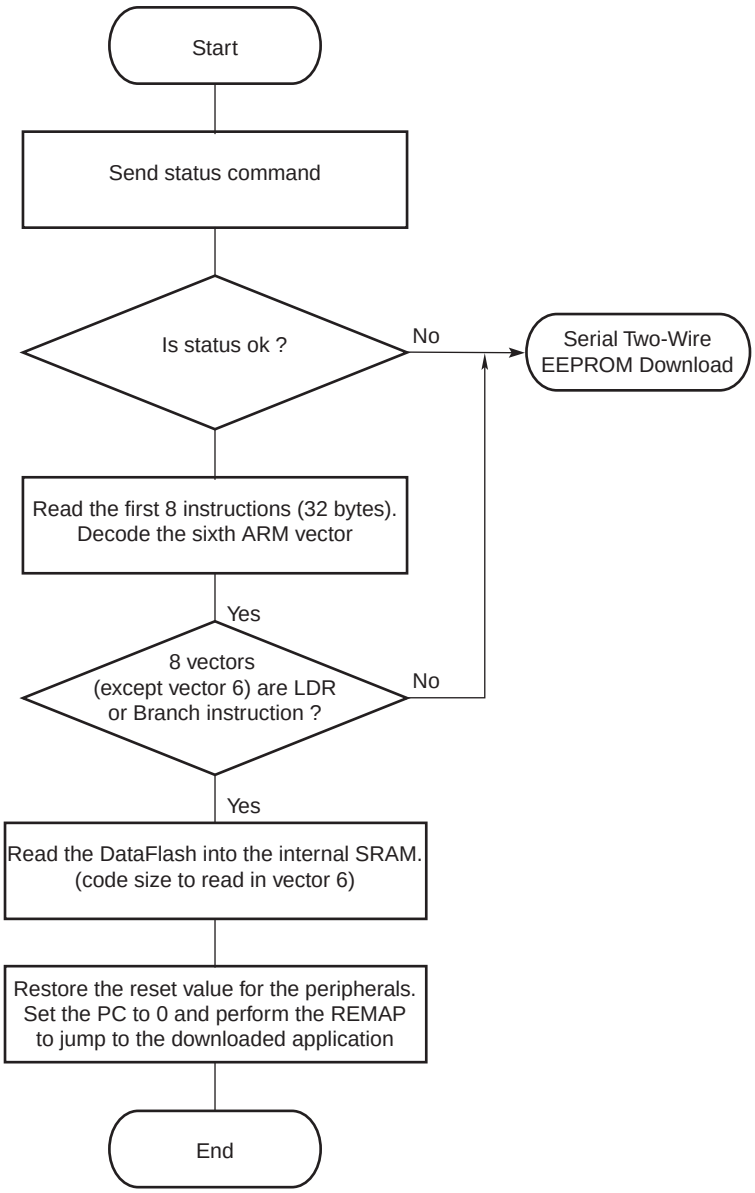
Serial DataFlash Download

The Boot Program supports all Atmel DataFlash devices. Table 12 summarizes the parameters to include in the ARM vector 6 for all devices.

The DataFlash has a Status Register that determines all the parameters required to access the device.

Thus, to be compatible with the future design of the DataFlash, parameters are coded in the ARM vector 6.

Figure 14. Serial DataFlash Download



Serial Two-wire EEPROM Download

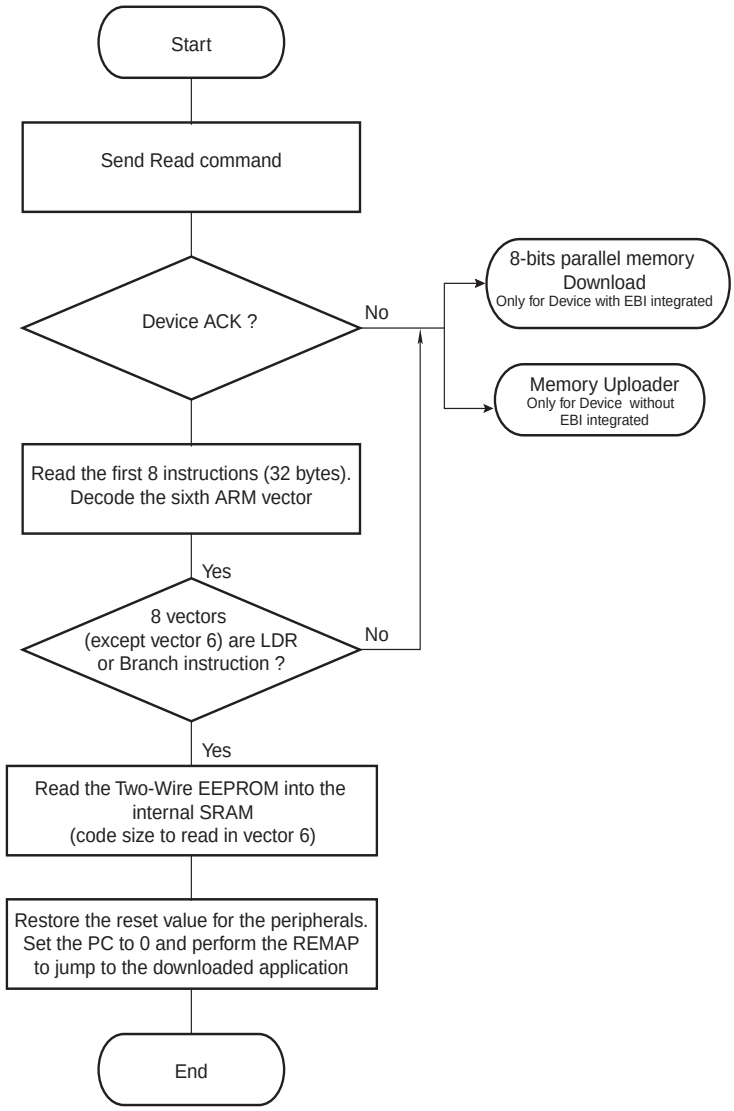
Generally, serial EEPROMs have no identification code. The bootloader checks for an acknowledgment on the first read. The device address on the two-wire bus must be 0x0.

The bootloader supports the devices listed in Table 14.

Table 14. Supported EEPROM Devices

Device	Size	Organization
AT24C16A	16 Kbits	16 bytes page write
AT24C164	16 Kbits	16 bytes page write
AT24C32	32 Kbits	32 bytes page write
AT24C64	64 Kbits	32 bytes page write
AT24C128	128 Kbits	64 bytes page write
AT24C256	256 Kbits	64 bytes page write
AT24C512	528 Kbits	128 bytes page write

Figure 15. Serial Two-Wire EEPROM Download

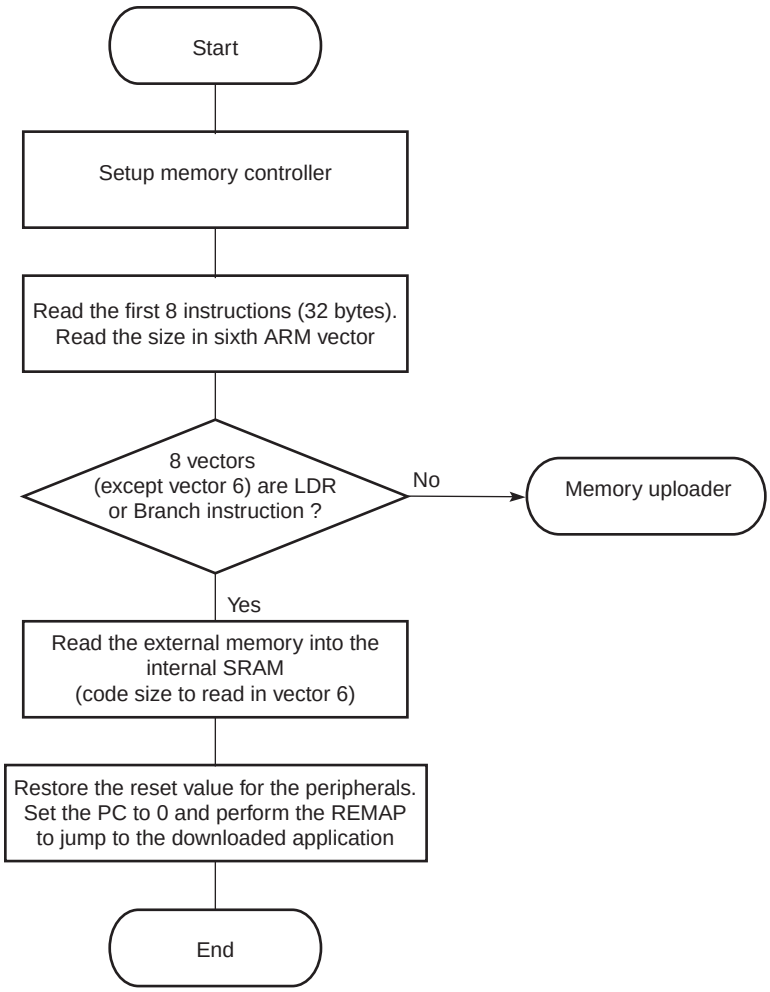


8-bit Parallel Flash Download (Only for Products Including an EBI)

Eight-bit parallel Flash download is supported if the product integrates an External Bus Interface (EBI).

All 8-bit memory devices supported by the EBI when NCS0 is configured in 8-bit data bus width are supported by the bootloader.

Figure 16. 8-bit Parallel Flash Download



Boot Uploader

If no valid boot device has been found during the Bootloader sequence, initialization of serial communication devices (DBGU and USB device ports) is performed.

- Initialization of the DBGU serial port (115200 bauds, 8, N, 1) and Xmodem protocol start
- Initialization of the USB Device Port and DFU protocol start
- Download of the application

The boot Uploader performs the DFU and Xmodem protocols to upload the application into internal SRAM at address 0x0020_0000.

The Boot Program uses a piece of internal SRAM for variables and stacks. To prevent any upload error, the size of the application to upload must be less than the SRAM size minus 3K bytes.

After the download, the peripheral registers are reset, the interrupts are disabled and the remap is performed. After the remap, the internal SRAM is at address 0x0000_0000 and the internal ROM at address 0x0010_0000. The instruction setting the PC to 0 is the one just after the remap command. This instruction is fetched in the pipe before doing the remap and executed just after. This fetch cycle executes the downloaded image.

External Communication Channels

DBGU Serial Port

The upload is performed through the DBGU serial port initialized to 115200 Baud, 8, n, 1.

The DBGU sends the character ‘C’ (0x43) to start an Xmodem protocol. Any terminal performing this protocol can be used to send the application file to the target. The size of the binary file to send depends on the SRAM size embedded in the product (Refer to the microcontroller datasheet to determine SRAM size embedded in the microcontroller). In all cases, the size of the binary file must be lower than SRAM size because the Xmodem protocol requires some SRAM memory to work.

Xmodem Protocol

The Xmodem protocol supported is the 128-byte length block. This protocol uses a two character CRC-16 to guarantee detection of a maximum bit error.

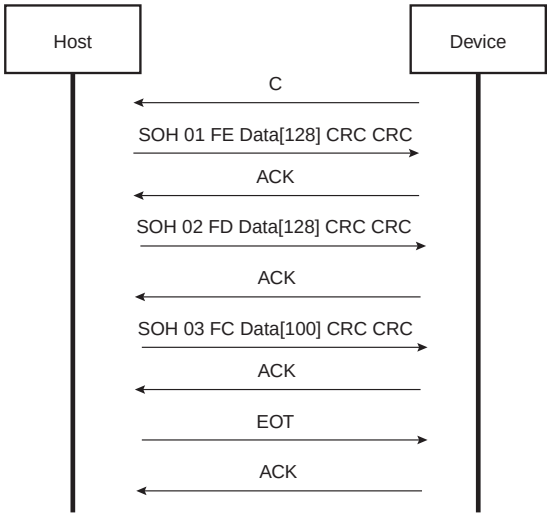
Xmodem protocol with CRC is accurate provided both sender and receiver report successful transmission. Each block of the transfer looks like:

<SOH><blk #><255-blk #><--128 data bytes--><checksum> in which:

- <SOH> = 01 hex
- <blk #> = binary number, starts at 01, increments by 1, and wraps 0FFH to 00H (not to 01)
- <255-blk #> = 1’s complement of the blk#.
- <checksum> = 2 bytes CRC16

Figure 17 shows a transmission using this protocol.

Figure 17. Xmodem Transfer Example



USB Device Port

A 48 MHz USB clock is necessary to use USB Device port. It has been programmed earlier in the device initialization with PLLB configuration.

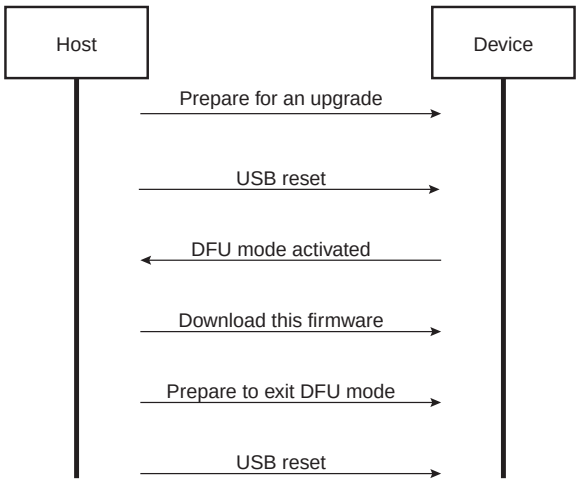
DFU Protocol

The DFU allows upgrade of the firmware of USB devices. The DFU algorithm is a part of the USB specification. For more details, refer to “USB Device Firmware Upgrade Specification, Rev. 1.0”.

There are four distinct steps when carrying out a firmware upgrade:

1. Enumeration: The device informs the host of its capabilities.
2. Reconfiguration: The host and the device agree to initiate a firmware upgrade.
3. Transfer: The host transfers the firmware image to the device. Status requests are employed to maintain synchronization between the host and the device.
4. Manifestation: Once the device reports to the host that it has completed the reprogramming operations, the host issues a reset and the device executes the upgraded firmware.

Figure 18. DFU Protocol



Hardware and Software Constraints

The software limitations of the Boot Program are:

- The downloaded code size is less than the SRAM size embedded in the product.
- The device address of the EEPROM must be 0 on the TWI bus.
- The code is always downloaded from the device address 0x0000_0000 (DataFlash, EEPROM) to the address 0x0000_0000 of the internal SRAM (after remap).
- The downloaded code must be position-independent or linked at address 0x0000_0000.

The hardware limitations of the Boot Program are:

- The DataFlash must be connected to NPCS0 of the SPI.
- The 8-bit parallel Flash must be connected to NCS0 of the EBI if the device integrates an EBI.

The SPI and TWI drivers use several PIOs in alternate functions to communicate with devices. Care must be taken when these PIOs are used by the application. The devices connected could be unintentionally driven at boot time, and electrical conflicts between SPI or TWI output pins and the connected devices may appear.

To assure correct functionality, it is recommended to plug in critical devices to other pins or to boot on an external 16-bit parallel memory (if product integrates an EBI) by setting bit BMS.

Table 15 contains a list of pins that are driven during the Boot Program execution. These pins are driven during the boot sequence for a period of about 6 ms if no correct boot program is found. The download through the TWI takes about 5 sec for 64K bytes due to the TWI bit rate (100 Kbits/s).

For the DataFlash driven by SPCK signal at 12 MHz, the time to download 64K bytes is reduced to 66 ms.

Before performing the jump to the application in internal SRAM, all the PIOs and peripherals used in the Boot Program are set to their reset state.

Table 15. Pins Driven during Boot Program Execution

Pin Used	SPI (Dataflash)	TWI (EEPROM)
MOSI(1)	O	X
SPCK(1)	O	X
NPCS0(1)	O	X
TWD(1)	X	I/O
TWCK(1)	X	O

Note: 1. See “Peripheral Multiplexing on PIO Lines” on page 13.

Embedded Software Services

Overview

An embedded software service is an independent software object that drives device resources for frequently implemented tasks. The object-oriented approach of the software provides an easy way to access services to build applications.

An AT91 service has several purposes:

- It gives software examples dedicated to the AT91 devices.
- It can be used on several AT91 device families.
- It offers an interface to the software stored in the ROM.

The main features of the software services are:

- Compliant with ATPCS
- Compliant with ANSI/ISO Standard C
- Compiled in ARM/Thumb Interworking
- ROM Entry Service
- Tempo, Xmodem and DataFlash services
- CRC and Sine tables

Service Definition

Service Structure

Structure Definition

A service structure is defined in C header files.

This structure is composed of data members and pointers to functions (methods) and is similar to a class definition. There is no protection of data access or methods access. However, some functions can be used by the customer application or other services and so be considered as public methods. Similarly, other functions are not invoked by them. They can be considered as private methods. This is also valid for data.

Methods

In the service structure, pointers to functions are supposed to be initialized by default to the standard functions. Only the default standard functions reside in ROM. Default methods can be overloaded by custom application methods.

Methods do not declare any static variables nor invoke global variables. All methods are invoked with a pointer to the service structure. A method can access and update service data without restrictions.

Similarly, there is no polling in the methods. In fact, there is a method to start the functionality (a read to give an example), a method to get the status (is the read achieved?), and a call-back, initialized by the start method. Thus, using service, the client application carries out a synchronous read by starting the read and polling the status, or an asynchronous read specifying a callback when starting the read operation.

Service Entry Point

Each AT91 service, except for the ***ROM Entry Service*** (see Section), defines a function named `AT91F_Open_<Service>`. It is the only entry point defined for a service. Even if the functions `AT91F_Open_<Service>` may be compared with object constructors, they do not act as constructors in that they initiate the service structure but they do not allocate it. ***Thus the customer application must allocate it.***

Example

```
// Allocation of the service structure
AT91S_Pipe pipe;
// Opening of the service
AT91PS_Pipe pPipe = AT91F_OpenPipe(&pipe, …);
```

Method pointers in the service structure are initialized to the default methods defined in the AT91 service. Other fields in the service structure are initialized to default values or with the arguments of the function AT91F_Open_<Service>.

In summary, an application must know what the service structure is and where the function AT91F_Open_<Service> is.

The default function AT91F_Open_<Service> may be redefined by the application or comprised in an application-defined function. See Section .

Using a Service

Opening a Service

The entry point to a service is established by initializing the service structure. An open function is associated with each service structure, except for the ***ROM Entry Service*** (see Section). Thus, only the functions AT91F_Open_<service> are visible from the user side. Access to the service methods is made via function pointers in the service structure.

The function AT91F_Open_<service> has at least one argument: a pointer to the service structure ***that must be allocated elsewhere***. It returns a pointer to the base service structure or a pointer to this service structure.

The function AT91F_Open_<service> initializes all data members and method pointers. All function pointers in the service structure are set to the service's functions.

The advantage of this method is to offer a single entry point for a service. The methods of a service are initialized by the open function and each member can be overloaded.

Overloading a Method

Default methods are defined for all services provided in ROM. These methods may not be adapted to a project requirement. It is possible to overload default methods by methods defined in the project.

A method is a pointer to a function. This pointer is initialized by the function AT91F_Open_<Service>. To overload one or several methods in a service, the function pointer must be updated to the new method.

It is possible to overload just one method of a service or all the methods of a service. In this latter case, the functionality of the service is user-defined, but still works on the same data structure.

Note: Calling the default function AT91F_Open_<Service> ensures that all methods and data are initialized.

This can be done by writing a new function My_OpenService(). This new Open function must call the library-defined function AT91F_Open_<Service>, and then update one or several function pointers:

Table 16. Overloading a Method with the Overloading of the Open Service Function

Default service behavior in ROM	Overloading AT91F_ChildMethod by My_ChildMethod

`// Defined in embedded_services.h`
`typedef struct _AT91S_Service {`
`  char data;`
`  char (*MainMethod) ();`
`  char (*ChildMethod) ();`
`} AT91S_Service, * AT91PS_Service;`
 	
`// Defined in obj_service.c (in ROM)`
`char AT91F_MainMethod ()`
`{`
`}`
 	
`char AT91F_ChildMethod ()`
`{`
`}`
 	
`// Init the service with default methods`
`AT91PS_Service AT91F_OpenService(`
`AT91PS_Service pService)`
`{`
`  pService->data = 0;`
`  pService->MainMethod =AT91F_MainMethod;`
`  pService->ChildMethod=AT91F_ChildMethod;`
`  return pService;`
`}`
	`// My_ChildMethod will replace AT91F_ChildMethod`
	`char My_ChildMethod ()`
	`{`
	`}`
	
	`// Overloading Open Service Method`
	`AT91PS_Service My_OpenService(`
	`AT91PS_Service pService)`
	`{`
	`  AT91F_OpenService(pService);`
	
	`// Overloading ChildMethod default value`
	`  pService->ChildMethod= My_ChildMethod;`
	`  return pService;`
	`}`
	
	`// Allocation of the service structure`
	`AT91S_Service service;`
	
	`// Opening of the service`
	`AT91PS_Service pService =`
	`My_OpenService(&service);`

This also can be done directly by overloading the method after the use of AT91F_Open_<Service> method:

Table 17. Overloading a Method without the Overloading of the Open Service Function.

Default service behavior in ROM	Overloading AT91F_ChildMethod by My_ChildMethod

`// Defined in embedded_services.h`
`typedef struct _AT91S_Service {`
`  char data;`
`  char (*MainMethod) ();`
`  char (*ChildMethod) ();`
`} AT91S_Service, * AT91PS_Service;`
 	
`// Defined in obj_service.c (in ROM)`
`char AT91F_MainMethod ()`
`{`
`}`
 	
`char AT91F_ChildMethod ()`
`{`
`}`
`// Init the service with default methods`
`AT91PS_Service AT91F_OpenService(`
`AT91PS_Service pService)`
`{`
`  pService->data = 0;`
`  pService->MainMethod =AT91F_MainMethod;`
`  pService->ChildMethod=AT91F_ChildMethod;`
`  return pService;`
`}`
	`// My_ChildMethod will replace AT91F_ChildMethod`
	`char My_ChildMethod ()`
	`{`
	`}`
	
	`// Allocation of the service structure`
	`AT91S_Service service;`
	
	`// Opening of the service`
	`AT91PS_Service pService =`
	`AT91F_OpenService(&service);`
	
	`// Overloading ChildMethod default value`
	`pService->ChildMethod= My_ChildMethod;`

Embedded Software Services

Definition

Several AT91 products embed ROM. In most cases, the ROM integrates a bootloader and several services that may speed up the application and reduce the application code size.

When software is fixed in the ROM, the address of each object (function, constant, table, etc.) must be related to a customer application. This is done by providing an address table to the linker. For each version of ROM, a new address table must be provided and all client applications must be recompiled.

The Embedded Software Services offer another solution to access objects stored in ROM. For each embedded service, the customer application requires only the address of the Service Entry Point (see Section).

Even if these services have only one entry point (`AT91F_Open_<Service>` function), they must be specified to the linker. The Embedded Software Services solve this problem by providing a dedicated service: the ***ROM Entry Service***.

The goal of this product-dedicated service is to provide just one address to access all ROM functionalities.

ROM Entry Service

The ***ROM Entry Service*** of a product is a structure named `AT91S_RomBoot`. Some members of this structure point to the open functions of all services stored in ROM (function `AT91F_Open_<Service>`) but also the CRC and Sine Arrays. Thus, only the address of the `AT91S_RomBoot` has to be published.

Table 18. Initialization of the ***ROM Entry Service*** and Use with an Open Service Method

Application Memory Space	ROM Memory Space

`// Init the ROM Entry Service`
`AT91S_RomBoot const *pAT91;`
`pAT91 = AT91C_ROM_BOOT_ADDRESS;`
 	
`// Allocation of the service structure`
`AT91S_CtlTempo tempo;`
 	
`// Call the Service Open method`
`pAT91->OpenCtlTempo(&tempo, ...);`
 	
`// Use of tempo methods`
`tempo.CtlTempoCreate(&tempo, ...);`
	`AT91S_TempoStatus AT91F_OpenCtlTempo(`
	`  AT91PS_CtlTempo pCtlTempo,`
	`  void const *pTempoTimer )`
	`{`
	
	`...`
	`}`
	
	`AT91S_TempoStatus AT91F_CtlTempoCreate (`
	`  AT91PS_CtlTempo pCtrl,`
	`  AT91PS_SvcTempo pTempo)`
	`{`
	`...`
	
	`}`

The application obtains the address of the ***ROM Entry Service*** and initializes an instance of the `AT91S_RomBoot` structure. To obtain the Open Service Method of another service stored in ROM, the application uses the appropriate member of the `AT91S_RomBoot` structure.

The address of the `AT91S_RomBoot` can be found at the beginning of the ROM, after the exception vectors.

Tempo Service

Presentation

The ***Tempo Service*** allows a single hardware system timer to support several software timers running concurrently. This works as an object notifier.

There are two objects defined to control the ***Tempo Service***: `AT91S_CtlTempo` and `AT91S_SvcTempo`.

The application declares one instance of `AT91S_CtlTempo` associated with the hardware system timer. Additionally, it controls a list of instances of `AT91S_SvcTempo`.

Each time the application requires another timer, it asks the `AT91S_CtlTempo` to create a new instance of `AT91S_SvcTempo`, then the application initializes all the settings of `AT91S_SvcTempo`.

Tempo Service Description

Table 19. Tempo Service Methods

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`pAT91->`**`OpenCtlTempo`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_OpenCtlTempo(`
`AT91PS_CtlTempo pCtlTempo,`
`void const *pTempoTimer)`
	Member of `AT91S_RomBoot` structure.
	Corresponds to the Open Service Method for the Tempo Service.
	Input Parameters:
	Pointer on a Control Tempo Object.
	Pointer on a System Timer Descriptor Structure.
	Output Parameters:
	Returns 0 if OpenCtrlTempo successful.
	Returns 1 if not.

`// Typical Use:`
`AT91S_CtlTempo ctlTempo;`
`ctlTempo.`**`CtlTempoStart`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_STStart(void * pTimer)`
	Member of `AT91S_CtlTempo` structure.
	Start of the Hardware System Timer associated.
	Input Parameters:
	Pointer on a Void Parameter corresponding to a System Timer Descriptor Structure.
	Output Parameters:
	Returns 2.

`// Typical Use:`
`AT91S_CtlTempo ctlTempo;`
`ctlTempo.`**`CtlTempoIsStart`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_STIsStart(`
`AT91PS_CtlTempo pCtrl)`
	Member of `AT91S_CtlTempo` structure.
	Input Parameters:
	Pointer on a Control Tempo Object.
	Output Parameters:
	Returns the Status Register of the System Timer.

`// Typical Use:`
`AT91S_CtlTempo ctlTempo;`
`ctlTempo.`**`CtlTempoCreate`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_CtlTempoCreate (`
`AT91PS_CtlTempo pCtrl,`
`AT91PS_SvcTempo pTempo)`
	Member of `AT91S_CtlTempo` structure.
	Insert a software timer in the `AT91S_SvcTempo`'s list.
	Input Parameters:
	Pointer on a Control Tempo Object.
	Pointer on a Service Tempo Object to insert.
	Output Parameters:
	Returns 0 if the software tempo was created.
	Returns 1 if not.

Table 19. Tempo Service Methods (Continued)

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`AT91S_CtlTempo ctlTempo;`
`ctlTempo.`**`CtlTempoRemove`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_CtlTempoRemove`
`(AT91PS_CtlTempo pCtrl,`
`AT91PS_SvcTempo pTempo)`
	Member of `AT91S_CtlTempo` structure.
	Remove a software timer in the list.
	Input Parameters:
	Pointer on a Control Tempo Object.
	Pointer on a Service Tempo Object to remove.
	Output Parameters:
	Returns 0 if the tempo was created.
	Returns 1 if not.

`// Typical Use:`
`AT91S_CtlTempo ctlTempo;`
`ctlTempo.`**`CtlTempoTick`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_CtlTempoTick`
`(AT91PS_CtlTempo pCtrl)`
	Member of `AT91S_CtlTempo` structure.
	Refresh all the software timers in the list. Update their timeout and check if callbacks have to be launched. So, for example, this function has to be used when the hardware timer starts a new periodic interrupt if period interval timer is used.
	Input Parameters:
	Pointer on a Control Tempo Object.
	Output Parameters:
	Returns 1.

`// Typical Use:`
`AT91S_SvcTempo svcTempo;`
`svcTempo.`**`Start`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_SvcTempoStart (`
`AT91PS_SvcTempo pSvc,`
`unsigned int timeout,`
`unsigned int reload,`
`void (*callback) (AT91S_TempoStatus, void *),`
`void *pData)`
	Member of `AT91S_SvcTempo` structure.
	Start a software timer.
	Input Parameters:
	Pointer on a Service Tempo Object.
	Timeout to apply.
	Number of times to reload the tempo after timeout completed for periodic execution.
	Callback on a method to launch once the timeout completed.
	Allows to have a hook on the current service.
	Output Parameters:
	Returns 1.

`// Typical Use:`
`AT91S_SvcTempo svcTempo;`
`svcTempo.`**`Stop`**`(...);`
 	
`// Default Method:`
`AT91S_TempoStatus AT91F_SvcTempoStop (`
`AT91PS_SvcTempo pSvc)`
	Member of `AT91S_SvcTempo` structure.
	Force to stop a software timer.
	Input Parameters:
	Pointer on a Service Tempo Object.
	Output Parameters:
	Returns 1.

Note: `AT91S_TempoStatus` corresponds to an unsigned int.

Using the Service

The first step is to find the address of the open service method AT91F_OpenCtlTempo using the ***ROM Entry Service***.

Allocate one instance of AT91S_CtlTempo and AT91S_SvcTempo in the application memory space:

```
// Allocate the service and the control tempo
AT91S_CtlTempo ctlTempo;
AT91S_SvcTempo svcTempo1;
```

Initialize the AT91S_CtlTempo instance by calling the AT91F_OpenCtlTempo function:

```
// Initialize service
pAT91->OpenCtlTempo(&ctlTempo, (void *) &(pAT91->SYSTIMER_DESC));
```

At this stage, the application can use the AT91S_CtlTempo service members.

If the application wants to overload an object member, it can be done now. For example, if AT91F_CtlTempoCreate(&ctlTempo, &svcTempo1) method is to be replaced by the application defined as my_CtlTempoCreate(...), the procedure is as follows:

```
// Overload AT91F_CtlTempoCreate
ctlTempo.CtlTempoCreate = my_CtlTempoCreate;
```

In most cases, initialize the AT91S_SvcTempo object by calling the AT91F_CtlTempoCreate method of the AT91S_CtlTempo service:

```
// Init the svcTempo1, link it to the AT91S_CtlTempo object
ctlTempo.CtlTempoCreate(&ctlTempo, &svcTempo1);
```

Start the timeout by calling Start method of the svcTempo1 object. Depending on the function parameters, either a callback is started at the end of the countdown or the status of the timeout is checked by reading the TickTempo member of the svcTempo1 object.

```
// Start the timeout
svcTempo1.Start(&svcTempo1,100,0,NULL,NULL);
// Wait for the timeout of 100 (unity depends on the timer programmation)
// No repetition and no callback.
while (svcTempo1.TickTempo);
```

When the application needs another software timer to control a timeout, it:

- Allocates one instance of AT91S_SvcTempo in the application memory space

```
// Allocate the service
AT91S_SvcTempo svcTempo2;
```

- Initializes the AT91S_SvcTempo object calling the AT91F_CtlTempoCreate method of the AT91S_CtlTempo service:

```
// Init the svcTempo2, link it to the AT91S_CtlTempo object
ctlTempo.CtlTempoCreate(&ctlTempo, &svcTempo2);
```

Xmodem Service

Presentation

The Xmodem service is an application of the communication pipe abstract layer. This layer is media-independent (USART, USB, etc.) and gives entry points to carry out reads and writes on an abstract media, the pipe.

Communication Pipe Service

The pipe communication structure is a virtual structure that contains all the functions required to read and write a buffer, regardless of the communication media and the memory management.

The pipe structure defines:

- a pointer to a communication service structure `AT91PS_SvcComm`
- a pointer to a buffer manager structure `AT91PS_Buffer`
- pointers on read and write functions
- pointers to callback functions associated to the read and write functions

The following structure defines the pipe object:

```
typedef struct _AT91S_Pipe
{
  // A pipe is linked with a peripheral and a buffer
  AT91PS_SvcComm pSvcComm;
  AT91PS_Buffer  pBuffer;

  // Callback functions with their arguments
  void (*WriteCallback) (AT91S_PipeStatus, void *);
  void (*ReadCallback)  (AT91S_PipeStatus, void *);
  void *pPrivateReadData;
  void *pPrivateWriteData;

  // Pipe methods
  AT91S_PipeStatus (*Write) (
    struct _AT91S_Pipe   *pPipe,
    char const *         pData,
    unsigned int         size,
    void                 (*callback) (AT91S_PipeStatus, void *),
    void                 *privateData);
  AT91S_PipeStatus (*Read) (
    struct _AT91S_Pipe  *pPipe,
    char                *pData,
    unsigned int        size,
    void                (*callback) (AT91S_PipeStatus, void *),
    void                *privateData);
  AT91S_PipeStatus (*AbortWrite) (struct _AT91S_Pipe  *pPipe);
  AT91S_PipeStatus (*AbortRead) (struct _AT91S_Pipe *pPipe);
  AT91S_PipeStatus (*Reset) (struct _AT91S_Pipe *pPipe);
  char (*IsWritten) (struct _AT91S_Pipe *pPipe,char const *pVoid);
  char (*IsReceived) (struct _AT91S_Pipe *pPipe,char const *pVoid);
} AT91S_Pipe, *AT91PS_Pipe;
```

The Xmodem protocol implementation demonstrates how to use the communication pipe.

Description of the Buffer Structure

The AT91PS_Buffer is a pointer to the AT91S_Buffer structure manages the buffers. This structure embeds the following functions:

- pointers to functions that manage the read buffer
- pointers to functions that manage the write buffer

All the functions can be overloaded by the application to adapt buffer management.

A simple implementation of buffer management for the ***Xmodem Service*** is provided in the boot ROM source code.

```
typedef struct _AT91S_Buffer
{
  struct _AT91S_Pipe *pPipe;
  void *pChild;

  // Functions invoked by the pipe
  AT91S_BufferStatus (*SetRdBuffer)     (struct _AT91S_Buffer *pSBuffer, char
*pBuffer, unsigned int Size);
  AT91S_BufferStatus (*SetWrBuffer)    (struct _AT91S_Buffer *pSBuffer, char const
*pBuffer, unsigned int Size);
  AT91S_BufferStatus (*RstRdBuffer)     (struct _AT91S_Buffer *pSBuffer);
  AT91S_BufferStatus (*RstWrBuffer)     (struct _AT91S_Buffer *pSBuffer);
  char (*MsgWritten)      (struct _AT91S_Buffer *pSBuffer, char const *pBuffer);
  char (*MsgRead)         (struct _AT91S_Buffer *pSBuffer, char const *pBuffer);

  // Functions invoked by the peripheral
  AT91S_BufferStatus (*GetWrBuffer)    (struct _AT91S_Buffer *pSBuffer, char const
**pData, unsigned int *pSize);
  AT91S_BufferStatus (*GetRdBuffer)     (struct _AT91S_Buffer *pSBuffer, char
**pData, unsigned int *pSize);
  AT91S_BufferStatus (*EmptyWrBuffer)   (struct _AT91S_Buffer *pSBuffer, unsigned
int size);
  AT91S_BufferStatus (*FillRdBuffer)    (struct _AT91S_Buffer *pSBuffer, unsigned
int size);
  char (*IsWrEmpty)      (struct _AT91S_Buffer *pSBuffer);
  char (*IsRdFull)       (struct _AT91S_Buffer *pSBuffer);
} AT91S_Buffer, *AT91PS_Buffer;
```

Description of the SvcComm Structure

The SvcComm structure provides the interface between low-level functions and the pipe object.

It contains pointers of functions initialized to the lower level functions (e.g. SvcXmodem).

The ***Xmodem Service*** implementation gives an example of SvcComm use.

```
typedef struct _AT91S_Service
{
  // Methods:
  AT91S_SvcCommStatus (*Reset)  (struct _AT91S_Service *pService);
  AT91S_SvcCommStatus (*StartTx)(struct _AT91S_Service *pService);
  AT91S_SvcCommStatus (*StartRx)(struct _AT91S_Service *pService);
  AT91S_SvcCommStatus (*StopTx) (struct _AT91S_Service *pService);
  AT91S_SvcCommStatus (*StopRx) (struct _AT91S_Service *pService);
  char                (*TxReady)(struct _AT91S_Service *pService);
  char                (*RxReady)(struct _AT91S_Service *pService);
  // Data:
  struct _AT91S_Buffer *pBuffer; // Link to a buffer object
  void *pChild;
} AT91S_SvcComm, *AT91PS_SvcComm;
```

Description of the SvcXmodem Structure

The SvcXmodem service is a reusable implementation of the Xmodem protocol. It supports only the 128-byte packet format and provides read and write functions. The SvcXmodem structure defines:

- a pointer to a handler initialized to readHandler or writeHandler
- a pointer to a function that processes the xmodem packet crc
- a pointer to a function that checks the packet header
- a pointer to a function that checks data

With this structure, the Xmodem protocol can be used with all media (USART, USB, etc.). Only private methods may be overloaded to adapt the Xmodem protocol to a new media.

The default implementation of the Xmodem uses a USART to send and receive packets. Read and write functions implement peripheral data controller facilities to reduce interrupt overhead. It assumes the USART is initialized, the memory buffer allocated and the interrupts programmed.

A periodic timer is required by the service to manage timeouts and the periodic transmission of the character “C” (Refer to Xmodem protocol). This feature is provided by the ***Tempo Service.***

The following structure defines the ***Xmodem Service***:

```
typedef struct _AT91PS_SvcXmodem {

  // Public Methods:
  AT91S_SvcCommStatus (*Handler) (struct _AT91PS_SvcXmodem *, unsigned int);
  AT91S_SvcCommStatus (*StartTx) (struct _AT91PS_SvcXmodem *, unsigned int);
  AT91S_SvcCommStatus (*StopTx)  (struct _AT91PS_SvcXmodem *, unsigned int);

  // Private Methods:
  AT91S_SvcCommStatus (*ReadHandler)  (struct _AT91PS_SvcXmodem *, unsigned int
csr);
  AT91S_SvcCommStatus (*WriteHandler) (struct _AT91PS_SvcXmodem *, unsigned int
csr);
  unsigned short      (*GetCrc)       (char *ptr, unsigned int count);
  char                (*CheckHeader)  (unsigned char currentPacket, char *packet);
  char                (*CheckData)    (struct _AT91PS_SvcXmodem *);

  AT91S_SvcComm  parent;      // Base class
  AT91PS_USART pUsart;

  AT91S_SvcTempo tempo; // Link to a AT91S_Tempo object

  char          *pData;
  unsigned int  dataSize;        // = XMODEM_DATA_STX or XMODEM_DATA_SOH
  char          packetDesc[AT91C_XMODEM_PACKET_SIZE];
  unsigned char packetId;         // Current packet
  char          packetStatus;
  char          isPacketDesc;
  char          eot;            // end of transmition
} AT91S_SvcXmodem, *AT91PS_SvcXmodem
```

Xmodem Service Description

Table 20. Xmodem Service Methods

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`pAT91->`**`OpenSvcXmodem`**`(...);`
 	
`// Default Method:`
`AT91PS_SvcComm AT91F_OpenSvcXmodem(`
`  AT91PS_SvcXmodem pSvcXmodem,`
`  AT91PS_USART pUsart,`
`  AT91PS_CtlTempo  pCtlTempo)`
	Member of `AT91S_RomBoot` structure.
	Corresponds to the Open Service Method for the Xmodem Service.
	
	Input Parameters:
	Pointer on SvcXmodem structure.
	Pointer on a USART structure.
	Pointer on a CtlTempo structure.
	Output Parameters:
	Returns the Xmodem Service Pointer Structure.

`// Typical Use:`
`AT91S_SvcXmodem svcXmodem;`
`svcXmodem.`**`Handler`**`(...);`
 	
`// Default read handler:`
`AT91S_SvcCommStatus`
`AT91F_SvcXmodemReadHandler(AT91PS_SvcXmodem`
`pSvcXmodem, unsigned int csr)`
 	
`// Default write handler:`
`AT91S_SvcCommStatus`
`AT91F_SvcXmodemWriteHandler(AT91PS_SvcXmodem`
`pSvcXmodem, unsigned int csr)`
	Member of `AT91S_SvcXmodem` structure.
	interrupt handler for xmodem read or write functionnalities
	Input Parameters:
	Pointer on a Xmodem Service Structure.
	csr: usart channel status register .
	Output Parameters:
	Status for xmodem read or write.

Using the Service

The following steps show how to initialize and use the ***Xmodem Service*** in an application:

```
Variables definitions:
AT91S_RomBoot const *pAT91; // struct containing Openservice functions
AT91S_SBuffer   sXmBuffer; // Xmodem Buffer allocation
AT91S_SvcXmodem svcXmodem; // Xmodem service structure allocation
AT91S_Pipe      xmodemPipe;// xmodem pipe communication struct
AT91S_CtlTempo  ctlTempo; // Tempo struct
AT91PS_Buffer pXmBuffer; // Pointer on a buffer structure
AT91PS_SvcComm pSvcXmodem; // Pointer on a Media Structure
Initialisations
// Call Open methods:
pAT91 = AT91C_ROM_BOOT_ADDRESS;
// OpenCtlTempo on the system timer
pAT91->OpenCtlTempo(&ctlTempo, (void *) &(pAT91->SYSTIMER_DESC));
ctlTempo.CtlTempoStart((void *) &(pAT91->SYSTIMER_DESC));
// Xmodem buffer initialisation
pXmBuffer     = pAT91->OpenSBuffer(&sXmBuffer);
pSvcXmodem = pAT91->OpenSvcXmodem(&svcXmodem, AT91C_BASE_DBGU, &ctlTempo);
// Open communication pipe on the xmodem service
pAT91->OpenPipe(&xmodemPipe, pSvcXmodem, pXmBuffer);
// Init the DBGU peripheral
// Open PIO for DBGU
  AT91F_DBGU_CfgPIO();
  // Configure DBGU
  AT91F_US_Configure (
    (AT91PS_USART) AT91C_BASE_DBGU,     // DBGU base address
    MCK,             // Master Clock
    AT91C_US_ASYNC_MODE,        // mode Register to be programmed
    BAUDRATE ,              // baudrate to be programmed
    0);                   // timeguard to be programmed
  // Enable Transmitter
  AT91F_US_EnableTx((AT91PS_USART) AT91C_BASE_DBGU);
  // Enable Receiver
  AT91F_US_EnableRx((AT91PS_USART) AT91C_BASE_DBGU);
// Initialize the Interrupt for System Timer and DBGU (shared interrupt)
// Initialize the Interrupt Source 1 for SysTimer and DBGU
  AT91F_AIC_ConfigureIt(AT91C_BASE_AIC,
             AT91C_ID_SYS,
             AT91C_AIC_PRIOR_HIGHEST,
             AT91C_AIC_SRCTYPE_INT_LEVEL_SENSITIVE,
             AT91F_ASM_ST_DBGU_Handler);

  // Enable SysTimer and DBGU interrupt
  AT91F_AIC_EnableIt(AT91C_BASE_AIC, AT91C_ID_SYS);
xmodemPipe.Read(&xmodemPipe, (char *) BASE_LOAD_ADDRESS, MEMORY_SIZE,
XmodemProtocol, (void *) BASE_LOAD_ADDRESS);
```

DataFlash Service

Presentation

The ***DataFlash Service*** allows the Serial Peripheral Interface (SPI) to support several Serial DataFlash and DataFlash Cards for reading, programming and erasing operations.

This service is based on SPI interrupts that are managed by a specific handler. It also uses the corresponding PDC registers.

For more information on the commands available in the DataFlash Service, refer to the relevant DataFlash documentation.

DataFlash Service Description

Table 21. DataFlash Service Methods

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`pAT91->OpenSvcDataFlash(...);`
 	
`// Default Method:`
`AT91PS_SvcDataFlash AT91F_OpenSvcDataFlash (`
`const AT91PS_PMC pApmc,`
`AT91PS_SvcDataFlash pSvcDataFlash)`
	Member of `AT91S_RomBoot` structure.
	Corresponds to the Open Service Method for the DataFlash Service.
	Input Parameters:
	Pointer on a PMC Register Description Structure.
	Pointer on a DataFlash Service Structure.
	Output Parameters:
	Returns the DataFlash Service Pointer Structure.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.Handler(...);`
 	
`// Default Method:`
`void AT91F_DataFlashHandler(`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned int status)`
	Member of `AT91S_SvcDataFlash` structure.
	SPI Fixed Peripheral C interrupt handler.
	Input Parameters:
	Pointer on a DataFlash Service Structure.
	Status: corresponds to the interruptions detected and validated on SPI (SPI Status Register masked by SPI Mask Register).
	Has to be put in the Interrupt handler for SPI.
	Output Parameters:
	None.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.Status(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus`
`AT91F_DataFlashGetStatus(AT91PS_DataflashDesc pDesc)`
	Member of `AT91S_SvcDataFlash` structure.
	Read the status register of the DataFlash.
	Input Parameters:
	Pointer on a DataFlash Descriptor Structure (member of the service structure).
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.AbortCommand(...);`
 	
`// Default Method:`
`void`
`AT91F_DataFlashAbortCommand(AT91PS_DataflashDesc pDesc)`
	Member of `AT91S_SvcDataFlash` structure
	Allows to reset PDC & Interrupts.
	Input Parameters:
	Pointer on a DataFlash Descriptor Structure (member of the service structure).
	Output Parameters:
	None.

Table 21. DataFlash Service Methods (Continued)

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`PageRead`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus AT91F_DataFlashPageRead (`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned int src,`
`unsigned char *dataBuffer,`
`int sizeToRead )`
	Member of `AT91S_SvcDataFlash` structure
	Read a Page in DataFlash.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	DataFlash address.
	Data buffer destination pointer.
	Number of bytes to read.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash Ready.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`ContinuousRead`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus`
`AT91F_DataFlashContinuousRead (`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`int src,`
`unsigned char *dataBuffer,`
`int sizeToRead )`
	Member of `AT91S_SvcDataFlash` structure.
	Continuous Stream Read.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	DataFlash address.
	Data buffer destination pointer.
	Number of bytes to read.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`ReadBuffer`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus AT91F_DataFlashReadBuffer`
`(`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned char BufferCommand,`
`unsigned int bufferAddress,`
`unsigned char *dataBuffer,`
`int sizeToRead )`
	Member of `AT91S_SvcDataFlash` structure.
	Read the Internal DataFlash SRAM Buffer 1 or 2.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	Choose Internal DataFlash Buffer 1 or 2 command.
	DataFlash address.
	Data buffer destination pointer.
	Number of bytes to read.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.
	Returns 4 if DataFlash Bad Command.
	Returns 5 if DataFlash Bad Address.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`MainMemoryToBufferTransfert`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus`
`AT91F_MainMemoryToBufferTransfert(`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned char BufferCommand,`
`unsigned int page)`
	Member of `AT91S_SvcDataFlash` structure
	Read a Page in the Internal SRAM Buffer 1 or 2.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	Choose Internal DataFlash Buffer 1 or 2 command.
	Page to read.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.
	Returns 4 if DataFlash Bad Command.

Table 21. DataFlash Service Methods (Continued)

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`PagePgmBuf`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus`
`AT91F_DataFlashPagePgmBuf(`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned char BufferCommand,`
`unsigned char *src,`
`unsigned int dest,`
`unsigned int SizeToWrite)`
	Member of `AT91S_SvcDataFlash` structure
	Page Program through Internal SRAM Buffer 1 or 2.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	Choose Internal DataFlash Buffer 1 or 2 command.
	Source buffer.
	DataFlash destination address.
	Number of bytes to write.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.
	Returns 4 if DataFlash Bad Command.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`WriteBuffer`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus`
`AT91F_DataFlashWriteBuffer (`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned char BufferCommand,`
`unsigned char *dataBuffer,`
`unsigned int bufferAddress,`
`int SizeToWrite )`
	Member of `AT91S_SvcDataFlash` structure.
	Write data to the Internal SRAM buffer 1 or 2.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	Choose Internal DataFlash Buffer 1 or 2 command.
	Pointer on data buffer to write.
	Address in the internal buffer.
	Number of bytes to write.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.
	Returns 4 if DataFlash Bad Command.
	Returns 5 if DataFlash Bad Address.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`WriteBufferToMain`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus AT91F_WriteBufferToMain (`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned char BufferCommand,`
`unsigned int dest )`
	Member of `AT91S_SvcDataFlash` structure.
	Write Internal Buffer to the DataFlash Main Memory.
	Input Parameters:
	Pointer on DataFlash Service Structure.
	Choose Internal DataFlash Buffer 1 or 2 command.
	Main memory address on DataFlash.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash is Ready.

Table 21. DataFlash Service Methods (Continued)

Associated Function Pointers & Methods Used by Default	Description

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`PageErase`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus AT91F_PageErase (`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned int PageNumber)`
	Member of `AT91S_SvcDataFlash` structure.
	Erase a page in DataFlash.
	Input Parameters:
	Pointer on a Service DataFlash Object.
	Page to erase.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash Ready.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`BlockErase`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus AT91F_BlockErase (`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned int BlockNumber )`
	Member of `AT91S_SvcDataFlash` structure.
	Erase a block of 8 pages.
	Input Parameters:
	Pointer on a Service DataFlash Object.
	Block to erase.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash Ready.

`// Typical Use:`
`AT91S_SvcDataFlash svcDataFlash;`
`svcDataFlash.`**`MainMemoryToBufferCompare`**`(...);`
 	
`// Default Method:`
`AT91S_SvcDataFlashStatus`
`AT91F_MainMemoryToBufferCompare(`
`AT91PS_SvcDataFlash pSvcDataFlash,`
`unsigned char BufferCommand,`
`unsigned int page)`
	Member of `AT91S_SvcDataFlash` structure.
	Compare the contents of a Page and one of the Internal SRAM buffer.
	Input Parameters:
	Pointer on a Service DataFlash Object.
	Internal SRAM DataFlash Buffer to compare command.
	Page to compare.
	Output Parameters:
	Returns 0 if DataFlash is Busy.
	Returns 1 if DataFlash Ready.
	Returns 4 if DataFlash Bad Command.

Note: AT91S_SvcDataFlashStatus corresponds to an unsigned int.

Using the Service

The first step is to find the address of the open service method `AT91F_OpenSvcDataFlash` using the ***ROM Entry Service***.

1. Allocate one instance of `AT91S_SvcDataFlash` and `AT91S_Dataflash` in the application memory space:

```
// Allocate the service and a device structure.
AT91S_SvcDataFlash svcDataFlash;
AT91S_Dataflash Device; // member of AT91S_SvcDataFlash service
```

   Then initialize the `AT91S_SvcDataFlash` instance by calling the `AT91F_OpenSvcDataFlash` function:

```
// Initialize service
pAT91->OpenSvcDataFlash (AT91C_BASE_PMC, &svcDataFlash);
```

2. Initialize the `SPI Interrupt`:

```
// Initialize the SPI Interrupt
at91_irq_open ( AT91C_BASE_AIC,AT91C_ID_SPI,3,
                    AT91C_AIC_SRCTYPE_INT_LEVEL_SENSITIVE ,AT91F_spi_asm_handler);
```

3. Configure the DataFlash structure with its correct features and link it to the device structure in the `AT91S_SvcDataFlash` service structure:

```
// Example with an ATMEL AT45DB321B DataFlash
Device.pages_number = 8192;
Device.pages_size = 528;
Device.page_offset = 10;
Device.byte_mask = 0x300;
// Link to the service structure
svcDataFlash.pDevice = &Device;
```

4. Now the different methods can be used. Following is an example of a Page Read of 528 bytes on page 50:

```
// Result of the read operation in RxBufferDataFlash
unsigned char RxBufferDataFlash[528];
svcDataFlash.PageRead(&svcDataFlash,
       (50*528),RxBufferDataFlash,528);
```

CRC Service

Presentation This “service” differs from the preceding ones in that it is structured differently: it is composed of an array and some methods directly accessible via the `AT91S_RomBoot` structure.

CRC Service Description

Table 22. CRC Service Description

Methods and Array Available	Description

`// Typical Use:`
`pAT91->CRC32(...);`
 	
`// Default Method:`
`void CalculateCrc32(`
`const unsigned char *address,`
`unsigned int size,`
`unsigned int *crc)`
	This function provides a table driven 32bit CRC generation for byte data. This CRC is known as the CCITT CRC32.
	Input Parameters:
	Pointer on the data buffer.
	The size of this buffer.
	A pointer on the result of the CRC.
	Output Parameters:
	None.

`// Typical Use:`
`pAT91->CRC16(...);`
 	
`// Default Method:`
`void CalculateCrc16(`
`const unsigned char *address,`
`unsigned int size,`
`unsigned short *crc)`
	This function provides a table driven 16bit CRC generation for byte data. This CRC is calculated with the POLYNOME 0x8005
	Input Parameters:
	Pointer on the data buffer.
	The size of this buffer.
	A pointer on the result of the CRC.
	Output Parameters:
	None.

`// Typical Use:`
`pAT91->CRCHDLC(...);`
 	
`// Default Method:`
`void CalculateCrcHdlc(`
`const unsigned char *address,`
`unsigned int size,`
`unsigned short *crc)`
	This function provides a table driven 16bit CRC generation for byte data. This CRC is known as the HDLC CRC.
	Input Parameters:
	Pointer on the data buffer.
	The size of this buffer.
	A pointer on the result of the CRC.
	Output Parameters:
	None.

`// Typical Use:`
`pAT91->CRCCCITT(...);`
 	
`// Default Method:`
`void CalculateCrc16ccitt(`
`const unsigned char *address,`
`unsigned int size,`
`unsigned short *crc)`
	This function provides a table driven 16bit CRC generation for byte data. This CRC is known as the CCITT CRC16 (POLYNOME = 0x1021).
	Input Parameters:
	Pointer on the data buffer.
	The size of this buffer.
	A pointer on the result of the CRC.
	Output Parameters:
	None.

`// Typical Use:`
`char reverse_byte;`
`reverse_byte = pAT91->Bit_Reverse_Array[...];`
 	
`// Array Embedded:`
`const unsigned char bit_rev[256]`
	Bit Reverse Array: array which allows to reverse one octet. Frequently used in mathematical algorithms.
	
	Used for example in the CRC16 calculation.

Using the Service

Compute the CRC16 CCITT of a 256-byte buffer and save it in the crc16 variable:

```
// Compute CRC16 CCITT
unsigned char BufferToCompute[256];
short crc16;
... (BufferToCompute Treatment)
pAT91->CRCCCITT(&BufferToCompute,256,&crc16);
```

Sine Service

Presentation This “service” differs from the preceding one in that it is structured differently: it is composed of an array and a method directly accessible through the `AT91S_RomBoot` structure.

Sine Service Description

Table 23. Sine Service Description

Method and Array Available	Description

`// Typical Use:`
`pAT91->`**`Sine`**`(...);`
 	
`// Default Method:`
`short AT91F_Sinus(int step)`
	This function returns the amplitude coded on 16 bits, of a sine waveform for a given step.
	Input Parameters:
	Step of the sine. Corresponds to the precision of the amplitude calculation. Depends on the Sine Array used. Here, the array has 256 values (thus 256 steps) of amplitude for 180 degrees.
	Output Parameters:
	Amplitude of the sine waveform.

`// Typical Use:`
`short sinus;`
`sinus = pAT91->`**`SineTab`**`[...];`
 	
`// Array Embedded:`
`const short AT91C_SINUS180_TAB[256]`
	Sine Array with a resolution of 256 values for 180 degrees.

Reset Controller

Overview

The AT91RM3400 has one reset input line called NRST. This line provides:

- Initialization of the User Interface registers (defined in the user interface of each peripheral) and sampling of the signals needed at bootup. It forces the processor to fetch the next instruction at address zero.
- Initialization of the embedded ICE TAP controller.

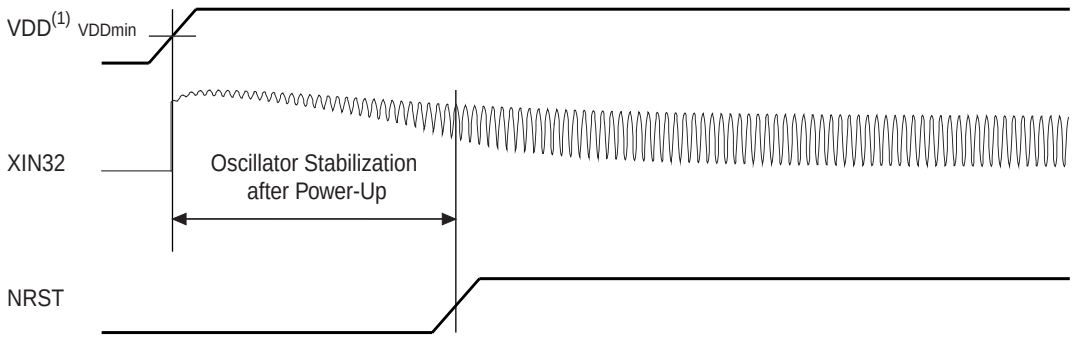
The NRST signal is considered as the System Reset signal and the reader must take care when designing the logic to drive this reset signal. It is an active low signal that asynchronously resets the logic in the AT91RM34000.

NRST Conditions

NRST is the active low reset input. When power is first applied to the system, a power-on reset (also called a “cold” reset) must be applied to the AT91RM3400. During this transient state, it is mandatory to hold the reset signal low long enough for the power supply to reach a working nominal level and for the oscillator to reach a stable operating frequency. Typically, these features are provided by all power supply supervisors with electrical characteristics considered as not nominal below a certain threshold voltage limit. Power-up is not the only event that must be considered; power-down or a brownout are also occurrences to assert the NRST signal. This threshold voltage must be selected according to the minimum operating voltage of the AT91RM3400 power supply lines marked as VDD in Figure 19. (See “DC Characteristics” on page 432.).

The choice of the reset holding delay depends on the start-up time of the low frequency oscillator as shown in Figure 19 (See “32 kHz Oscillator Characteristics” on page 435.).

Figure 19. Cold Reset and Oscillator Start-up Relationship



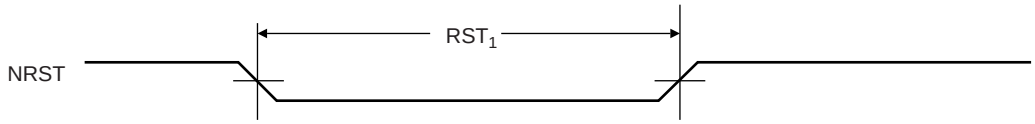
Note: 1. VDD is applicable to V_{DDIO}, V_{DDPLL}, V_{DDOSC} and V_{DDCORE}.

NRST can also be asserted in circumstances other than the power-up sequence, such as a manual command. In this case, assertion can be performed asynchronously, but exit from reset is synchronized internally to the default active clock. During normal operation, NRST must be active for a minimum delay time to ensure correct behavior (see Figure 20 and Table 24).

Table 24. Reset Minimum Pulse Width

Symbol	Parameter	Minimum Pulse Width	Unit
RST_1	NRST Minimum Pulse Width	92	µs

Figure 20. NRST Assertion



Reset Management

The system reset functionality is provided via the NRST signal.

The reset signal forces the microcontroller to assume a set of initial conditions:

- Default states (default value) of the user interface are restored.
- The processor is required to perform the next instruction fetch from address zero.

With the exception of the program counter and the Current Program Status Register, the processor's registers do not have defined reset states. When the microcontroller's NRST input is asserted, the processor immediately stops execution of the current instruction, independent of the clock.

The system reset circuitry must take two types of reset requests into account:

- Cold reset needed for the power-up sequence
- User reset request

Both have the same effect but can have different assertion time requirements regarding the NRST pin. In fact, the cold reset assertion has to overlap the start-up time of the system. The user reset request requires a smaller assertion delay time than the cold reset.

Recommended Features of the Reset Controller

The following table gives an overview of the recommended features of a reset controller in order to obtain an optimal system with the AT91RM3400 device.

Table 25. Reset Controller Function Overview

Feature	Description
Power Supply Monitoring	Overlaps the transient state of the system during power-up/down and brownout.
Reset Active Timeout Period	Overlaps the start-up time of the boot-up oscillator by holding the reset signal during this delay.
Manual Reset Command	Asserts the reset signal from a logic command and holds the reset signal with a shorter delay than the Reset Active Timeout Period.

Memory Controller (MC)

Overview

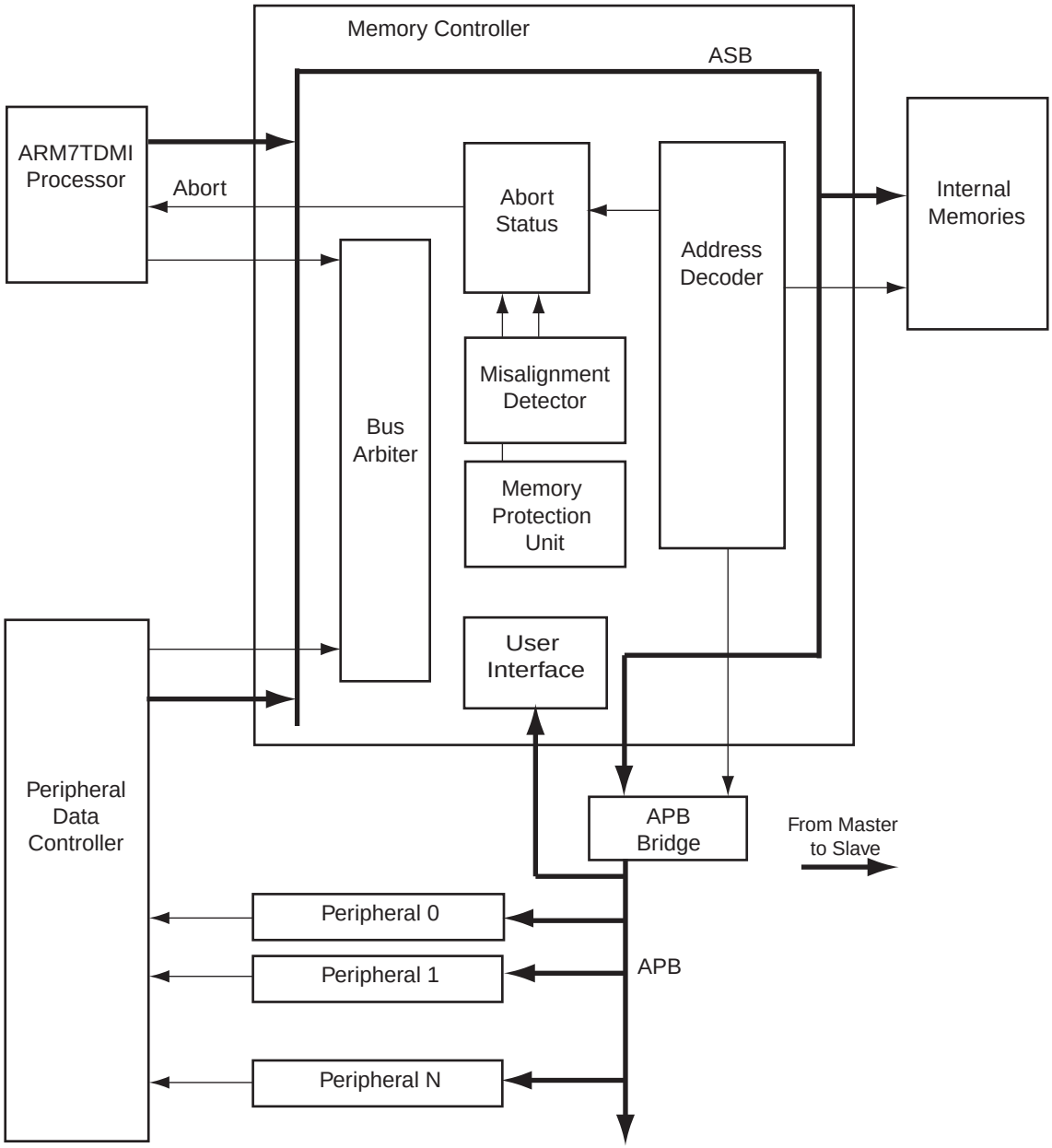
The Memory Controller (MC) manages the ASB bus and controls accesses requested by the masters, typically the ARM7TDMI processor and the Peripheral Data Controller. It features a simple bus arbiter, an address decoder, an abort status and a misalignment detector. In addition, the MC contains a Memory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

Main features of the AT91RM3400 Memory Controller are:

- Bus Arbiter
  - Handles Requests from the ARM7TDMI and the Peripheral Data Controller
- Address Decoder Provides Selection Signals for
  - Up to Four 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
  - Allows Remapping of an Internal SRAM in Place of the Internal ROM
  - Allows Handling of Dynamic Interrupt Vectors
- 16-area Memory Protection Unit
  - Individually Programmable Size Between 1K Bytes and 64M Bytes
  - Individually Programmable Protection Against Write and/or User Access
  - Peripheral Protection Against Write and/or User Access

Block Diagram

Figure 21. 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
- A memory protection unit

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.

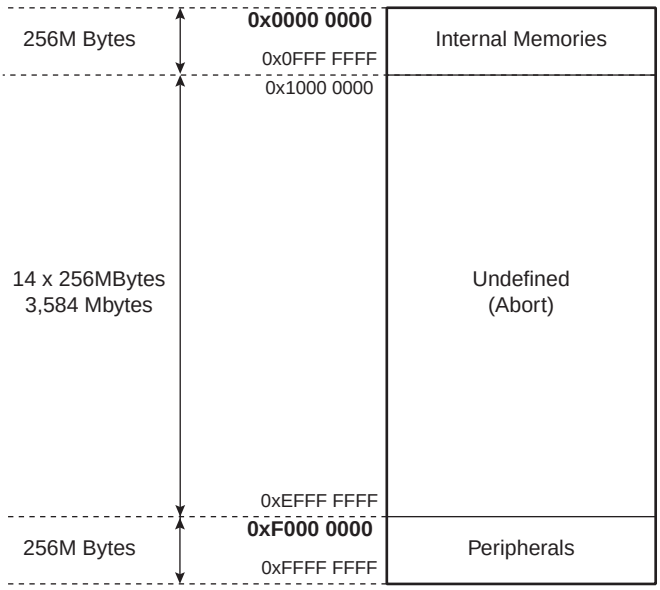
Address Decoder

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

Figure 22. Memory Areas



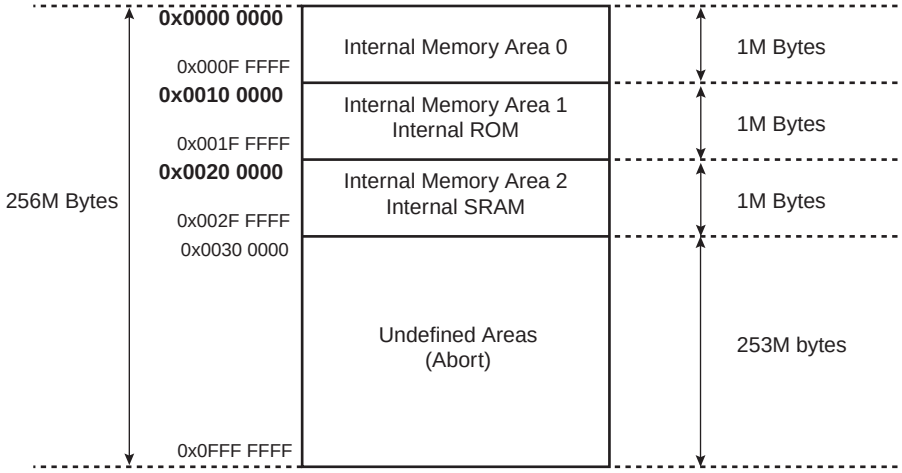
Internal Memory Mapping

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 23. 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 ROM is mapped into Internal Memory Area 0, so that the ARM7TDMI reaches an executable instruction contained in ROM. 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
- access to a protected area without the permitted state
- an access to a misaligned address.

When an abort occurs, a signal is sent back to all the masters, regardless of which one has generated the access. However, only the ARM7TDMI can take an abort signal into account, and only under the condition that it was generating an access. The Peripheral Data Controller does not handle the abort input signal. Note that the connection is not represented in Figure 21.

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

The full 32-bit wide abort address is saved in MC_AASR. Parameters of the access are saved in MC_ASR and include:

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

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

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

Memory Protection Unit

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

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

Programmming of the 16 memory spaces is done in the registers MC_PUIA0 to MC_PUIA15.

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

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

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

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

Misalignment Detector

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

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

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

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

AT91RM3400 Memory Controller (MC) User Interface

Base Address: 0xFFFFFF00

Table 26. 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	Reserved			
0x10	MC Protection Unit Area 0	MC_PUIA0	Read/Write	0x0
0x14	MC Protection Unit Area 1	MC_PUIA1	Read/Write	0x0
0x18	MC Protection Unit Area 2	MC_PUIA2	Read/Write	0x0
0x1C	MC Protection Unit Area 3	MC_PUIA3	Read/Write	0x0
0x20	MC Protection Unit Area 4	MC_PUIA4	Read/Write	0x0
0x24	MC Protection Unit Area 5	MC_PUIA5	Read/Write	0x0
0x28	MC Protection Unit Area 6	MC_PUIA6	Read/Write	0x0
0x2C	MC Protection Unit Area 7	MC_PUIA7	Read/Write	0x0
0x30	MC Protection Unit Area 8	MC_PUIA8	Read/Write	0x0
0x34	MC Protection Unit Area 9	MC_PUIA9	Read/Write	0x0
0x38	MC Protection Unit Area 10	MC_PUIA10	Read/Write	0x0
0x3C	MC Protection Unit Area 11	MC_PUIA11	Read/Write	0x0
0x40	MC Protection Unit Area 12	MC_PUIA12	Read/Write	0x0
0x44	MC Protection Unit Area 13	MC_PUIA13	Read/Write	0x0
0x48	MC Protection Unit Area 14	MC_PUIA14	Read/Write	0x0
0x4C	MC Protection Unit Area 15	MC_PUIA15	Read/Write	0x0
0x50	MC Protection Unit Peripherals	MC_PUP	Read/Write	0x0
0x54	MC Protection Unit Enable Register	MC_PUER	Read/Write	0x0

MC Remap Control Register

Register Name: MC_RCR

Access Type: Write-only

Absolute Address: 0xFFFF FF00

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

- **RCB: Remap Command Bit**

0: No effect.

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

MC Abort Status Register

Register Name: MC_ASR

Access Type: Read-only

Reset Value: 0x0

Absolute Address: 0xFFFF FF04

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

- **UNDADD: Undefined Address Abort Status**

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

1: The last abort was due to the access of an undefined address in the address space.

- **MISADD: Misaligned Address Abort Status**

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

1: The last aborted access was due to an address misalignment.

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

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

1: The last aborted access was due to the Memory Protection Unit.

- **ABTSZ: Abort Size Status .**

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

- **ABTTYP: Abort Type Status .**

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

- **MST0: 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

Absolute Address: 0xFFFF FF08

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

- **ABTADD: Abort Address**

This field contains the address of the last aborted access.

MC Protection Unit Area 0 to 15 Registers

Register Name: MC_PUIA0 - MC_PUIA15

Access Type: Read/Write

Reset Value: 0x0

Absolute Address: 0xFFFFFF10 - 0xFFFFFF4C

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

- **PROT: Protection :**

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

- **SIZE: Internal Area Size :**

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

- **BA: Internal Area Base Address**

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

MC Protection Unit Peripheral

Register Name: MC_PUP

Access Type: Read/Write

Reset Value: 0x000000000

Absolute Address: 0xFFFFFF50

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

- **PROT: Protection :**

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

MC Protection Unit Enable Register

Register Name: MC_PUER

Access Type: Read/Write

Reset Value: 0x000000000

Absolute Address: 0xFFFFFF54

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

- **PUEB: Protection Unit Enable Bit**

0: The Memory Controller Protection Unit is disabled.

1: The Memory Controller Protection Unit is enabled.

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 Contoller 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

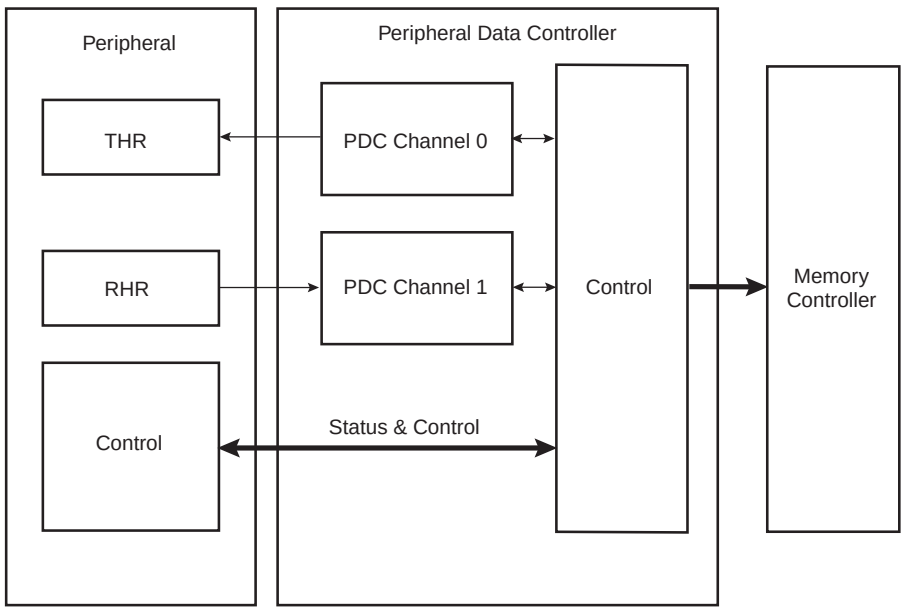
The peripheral triggers PDC transfers using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the corresponding peripheral.

Important features of the PDC are:

- Generates Transfers to/from Peripherals Such as DBGU, USART, SSC, SPI and MCI
- Supports Up to Twenty Channels (Product Dependent)
- One Master Clock Cycle Needed for a Transfer from Memory to Peripheral
- Two Master Clock Cycles Needed for a Transfer from Peripheral to Memory

Block Diagram

Figure 24. 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 transmitters requests.

Peripheral Data Controller (PDC) User Interface

Table 27. Register Mapping

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

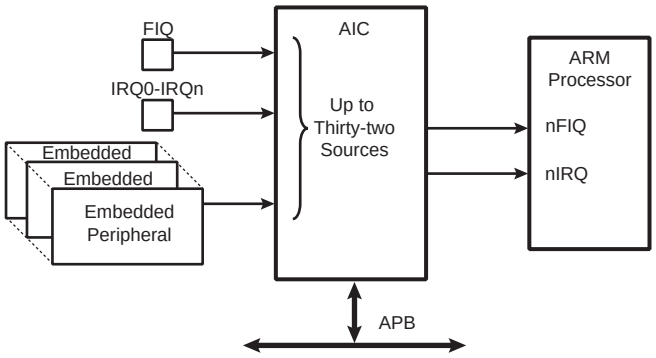
The fast forcing feature redirects any internal or external interrupt source to provide a fast interrupt rather than a normal interrupt.

Important Features of the AIC are:

- Controls the Interrupt Lines (nIRQ and nFIQ) of an ARM® Processor
- Thirty-two Individually Maskable and Vectored Interrupt Sources
  - Source 0 is Reserved for the Fast Interrupt Input (FIQ)
  - Source 1 is Reserved for System Peripherals (ST, RTC, PMC, DBGU…)
  - Source 2 to Source 31 Control up to Thirty Embedded 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 1 to 31
  - 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 when Protect ModeIs Are Enabled
- Fast Forcing
  - Permits Redirecting any Normal Interrupt Source on the Fast Interrupt of the Processor
- General Interrupt Mask
  - Provides Processor Synchronization on Events Without Triggering an Interrupt

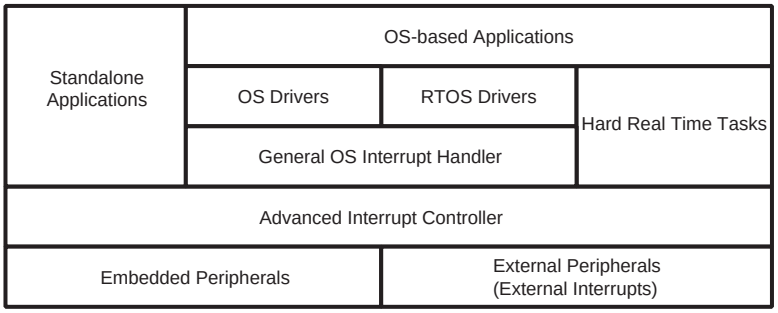
Block Diagram

Figure 25. Block Diagram



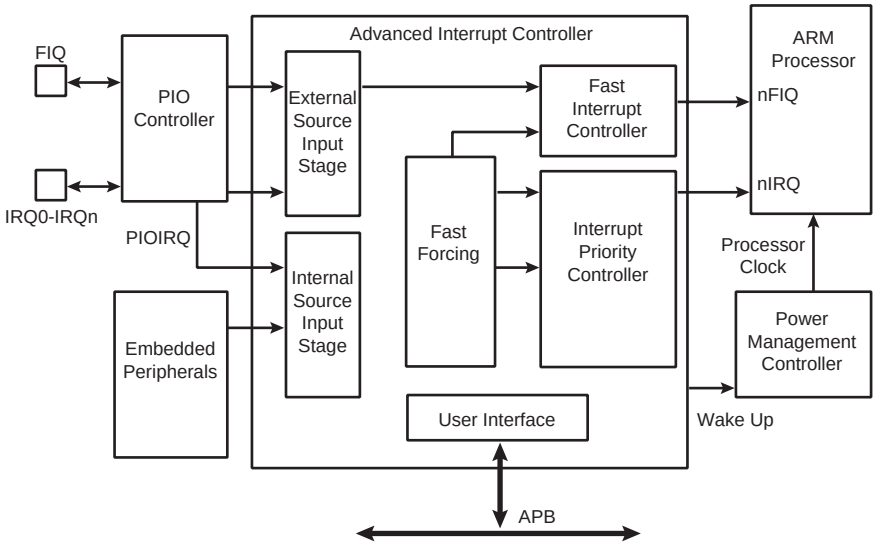
Application Block Diagram

Figure 26. Description of the Application Block



AIC Detailed Block Diagram

Figure 27. AIC Detailed Block Diagram



I/O Line Description

Table 28. 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 SRCTYPE 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 107.) 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 111.)

The automatic clear of the interrupt source 0 is performed when AIC_FVR is read.

Interrupt Status

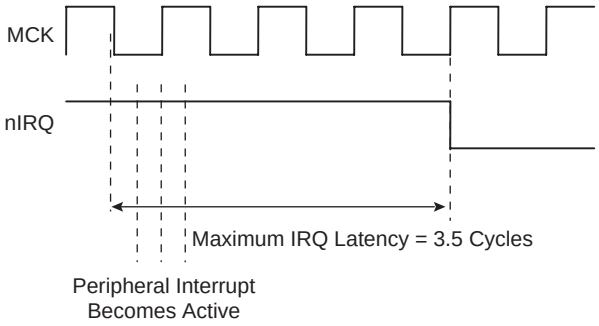
For each interrupt, the AIC operation originates in AIC_IPR (Interrupt Pending Register) and its mask in AIC_IMR (Interrupt Mask Register). AIC_IPR enables the actual activity of the sources, whether masked or not.

The AIC_ISR register reads the number of the current interrupt (see “Priority Controller” on page 107) 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.

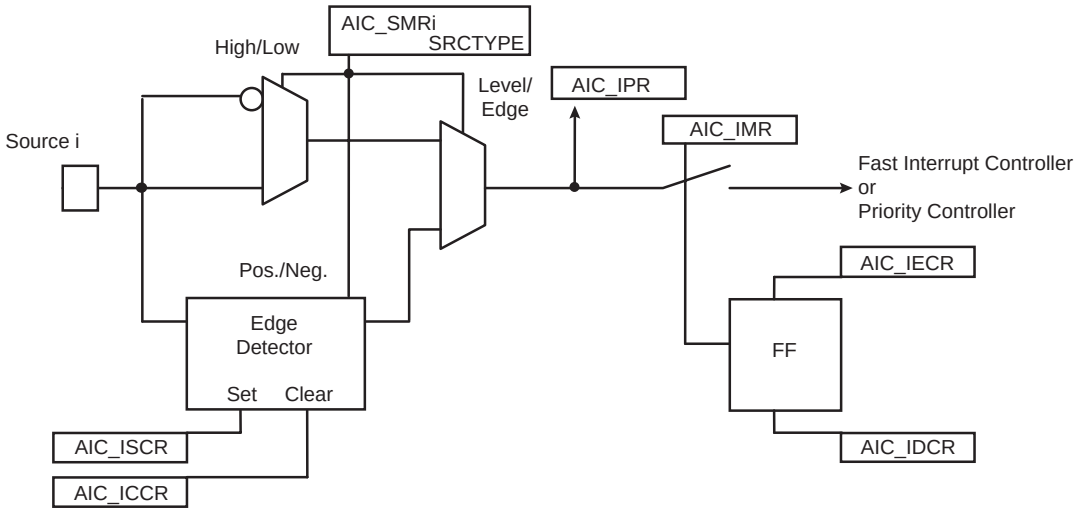
Internal Interrupt Source Input Stage

Figure 28. Internal Interrupt Source Input Stage



External Interrupt Source Input Stage

Figure 29. External Interrupt Source Input Stage



Interrupt Latencies

Global interrupt latencies depend on several parameters, including:

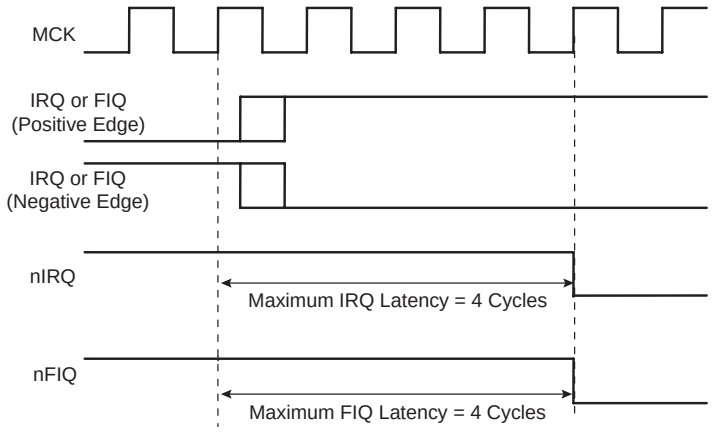
- The time the software masks the interrupts.
- Occurrence, either at the processor level or at the AIC level.
- The execution time of the instruction in progress when the interrupt occurs.
- The treatment of higher priority interrupts and the resynchronization of the hardware signals.

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.

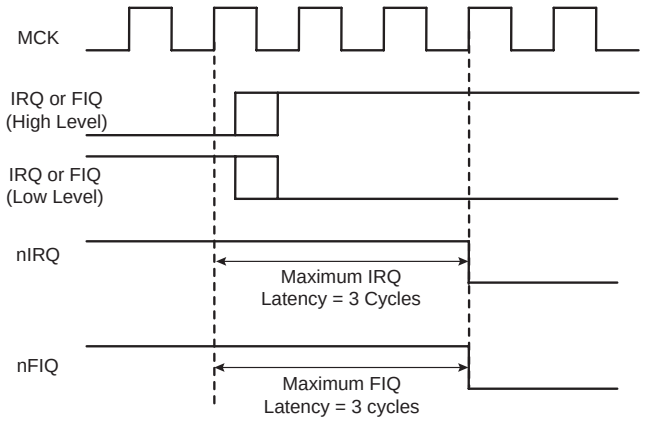
External Interrupt Edge Triggered Source

Figure 30. External Interrupt Edge Triggered Source



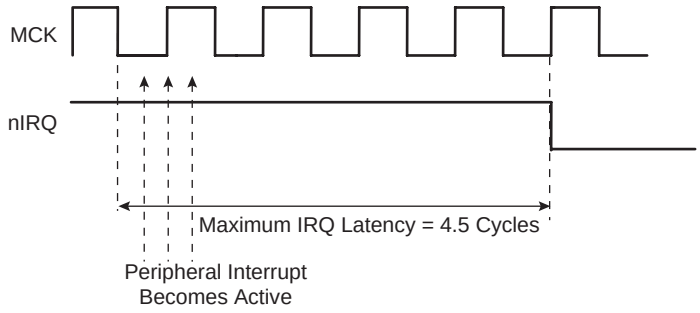
External Interrupt Level Sensitive Source

Figure 31. External Interrupt Level Sensitive Source



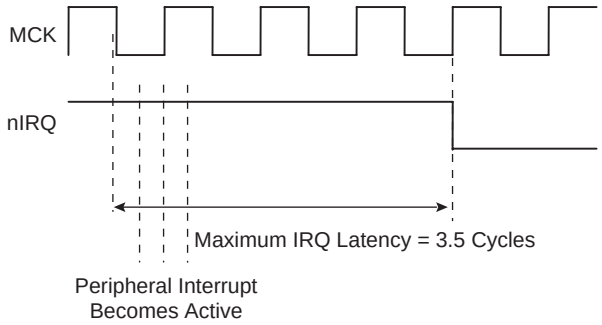
Internal Interrupt Edge Triggered Source

Figure 32. Internal Interrupt Edge Triggered Source



Internal Interrupt Level Sensitive Source

Figure 33. 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 highest 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 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_IECR (Interrupt Enable Command Register) and AIC_IDCR (Interrupt Disable Command Register) respectively enables and disables the fast interrupt. The bit 0 of AIC_IMR (Interrupt Mask Register) indicates whether the fast interrupt is enabled or disabled.

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_FFSR) that controls the feature for each internal or external interrupt source.

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

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

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

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

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

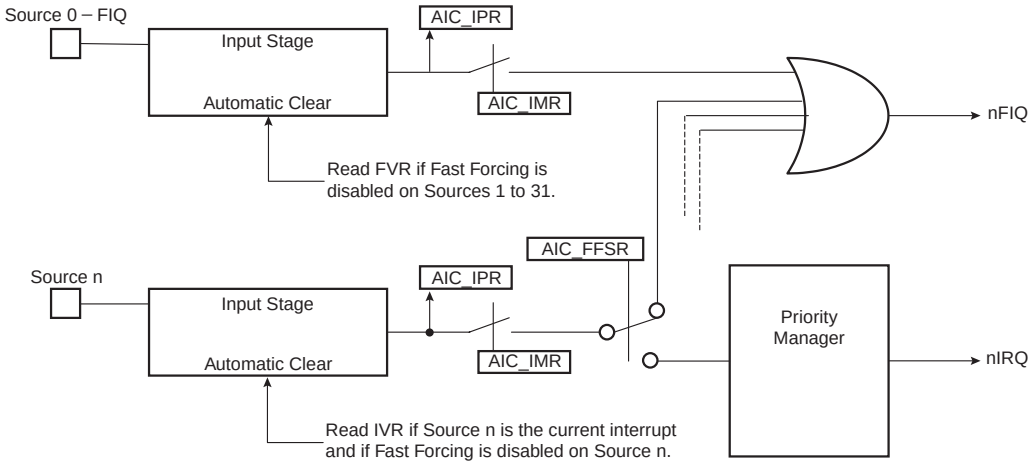
The Fast Interrupt Vector Register (AIC_FVR) reads the contents of the Source Vector Register 0 (AIC_SVR0), whatever the source of the fast interrupt may be. The read of the FVR does not clear the Source 0 when the fast forcing feature is used and the interrupt source should be cleared by writing to 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 34. 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 supports only an ± 4-Kbyte offset.

Table 29. 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[1]
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 the Interrupt Pending 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		

- **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 no 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	NIFQ

- **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 use 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 enable on the corresponding interrupt.

Power Management Controller (PMC)

Overview

The Power Management Controller (PMC) generates all the system clocks thanks to the integration of two oscillators and two PLLs.

The PMC provides clocks to the embedded processor and enables the idle mode by stopping the processor clock until the next interrupt.

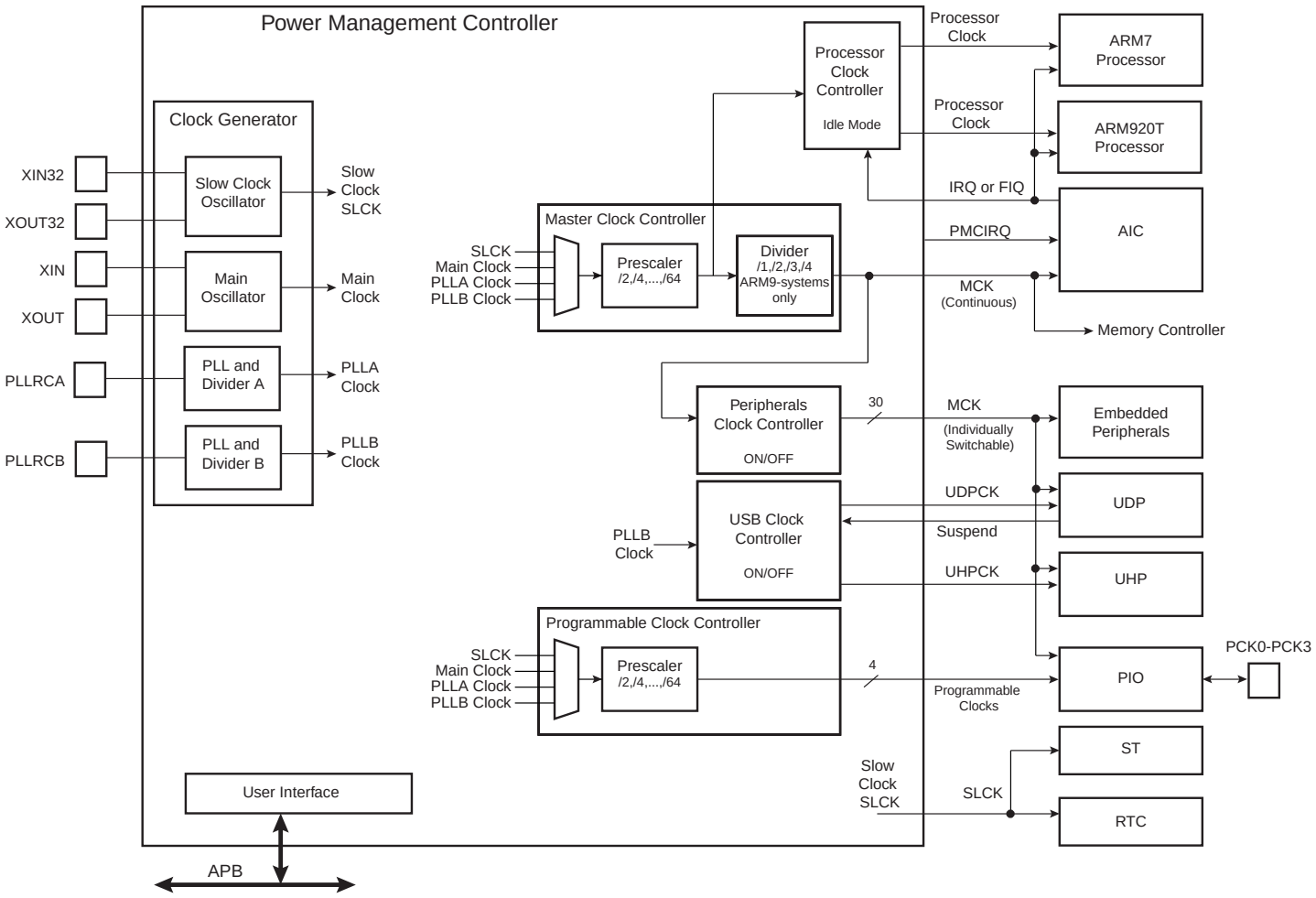
The PMC independently provides and controls up to thirty peripheral clocks and four programmable clocks that can be used as outputs on pins to feed external devices. The integration of the PLLs supplies the USB devices and host ports with a 48 MHz clock, as required by the bus speed, and the rest of the system with a clock at another frequency. Thus, the fully-featured Power Management Controller optimizes power consumption of the whole system and supports the Normal, Idle, Slow Clock and Standby operating modes.

The main features of the PMC are:

- Optimizes the Power Consumption of the Whole System
- Embeds and Controls
  - One Main Oscillator and One Slow Clock Oscillator (32.768 kHz)
  - Two Phase Locked Loops (PLLs) and Dividers
  - Clock Prescalers
- Provides
  - the Processor Clock PCK
  - the Master Clock MCK
  - up to two USB Clocks (depending on the USB Ports embedded)
    - UHPCK for the USB Host Port
    - UDPCK for the USB Device Port
  - Programmable Automatic PLL Switch-off in USB Device Suspend Conditions
  - up to Thirty Peripheral Clocks
  - up to Four Programmable Clock Outputs
- Four Operating Modes
  - Normal Mode, Idle Mode, Slow Clock Mode, Standby Mode

Block Diagram

Figure 35. Power Management Controller Block Diagram



Product Dependencies

I/O Lines

The Power Management Controller is capable of handling up to four Programmable Clocks, PCK0 to PCK3.

A Programmable Clock is generally multiplexed on a PIO Controller. The user must first program the PIO controllers to assign the pins of the Programmable Clock to its peripheral function.

Interrupt

The Power Management Controller has an interrupt line connected to the Advanced Interrupt Controller (AIC). Handling the PMC interrupt requires programming the AIC before configuring the PMC.

Oscillator and PLL Characteristics

The electrical characteristics of the embedded oscillators and PLLs are product-dependent, even if the way to control them is similar.

All of the parameters for both oscillators and the PLLs are given in the DC Characteristics section of the product datasheet. These figures are used not only for the hardware design, as they affect the external components to be connected to the pins, but also the software configuration, as they determine the waiting time for the startup and lock times to be programmed.

Peripheral Clocks

The Power Management Controller provides and controls up to thirty peripheral clocks. The bit number permitting the control of a peripheral clock is the Peripheral ID of the embedded peripheral.

When the Peripheral ID does not correspond to a peripheral, either because this is an external interrupt or because there are less than thirty peripherals, the control bits of the Peripheral ID are not implemented in the PMC and programming them has no effect on the behavior of the PMC.

USB Clocks

The Power Management Controller provides and controls two USB Clocks, the UHPCK for the USB Host Port, and the UDPCK for the USB Device.

If the product does not embed the USB Host Port or the USB Device Port, the associated control bits and registers are not implemented in the PMC and programming them has no effect on the behavior of the PMC.

Functional Description

Operating Modes Definition

The following operating modes are supported by the PMC and offer different power consumption levels and event response latency times:

- Normal Mode: The ARM processor clock is enabled and peripheral clocks are enabled depending on application requirements.
- Idle Mode: The ARM processor clock is disabled and waiting for the next interrupt (or a main reset). The peripheral clocks are enabled depending on application requirements. PDC transfers are still possible.
- Slow Clock Mode: Slow clock mode is similar to normal mode, but the main oscillator and the PLL are switched off to save power and the processor and the peripherals run in Slow Clock mode. Note that slow clock mode is the mode selected after the reset.
- Standby Mode: Standby mode is a combination of Slow Clock mode and Idle Mode. It enables the processor to respond quickly to a wake-up event by keeping power consumption very low.

Clock Definitions

The Power Management Controller provides the following clocks:

- Slow Clock (SLCK), typically at 32.768 kHz, is the only permanent clock within the system.
- Master Clock (MCK), 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), typically the Master Clock for ARM7-based systems and a faster clock on ARM9-based systems, switched off when entering idle mode.
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SSC, SPI, TWI, TC, MCI, etc.) and independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- UDP Clock (UDPCK), typically at 48 MHz, required by the USB Device Port operations.
- UHP Clock (UHPCK), typically at 48 MHz, required by the USB Host Port operations.
- Programmable Clock Outputs (PCK0 to PCK3) can be selected from the clocks provided by the clock generator and driven on the PCK0 to PCK3 pins.

Clock Generator

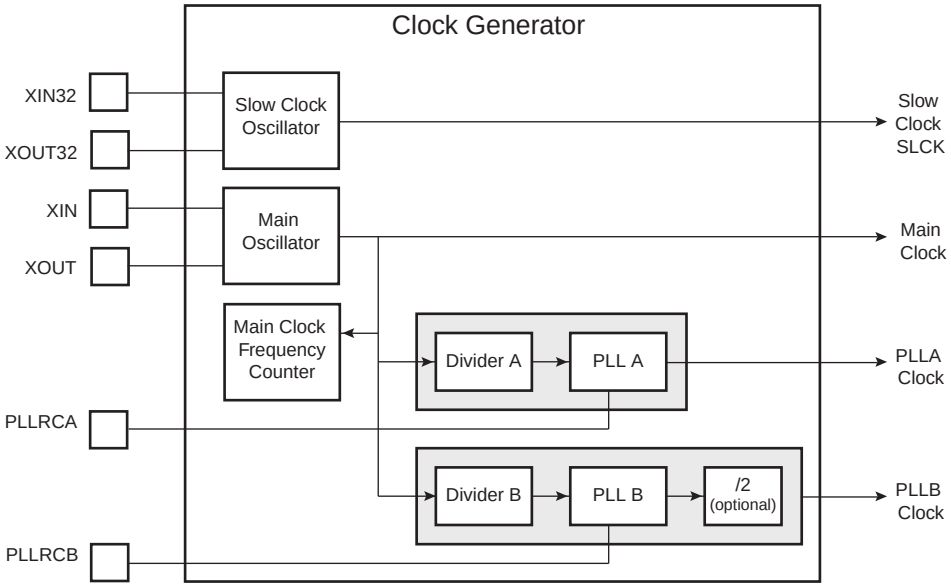
The Clock Generator embeds:

- the Slow Clock Oscillator
- the Main Oscillator
- two PLL and divider blocks, A and B

The Clock Generator integrates as an option a divider by 2. The ARM7-based systems generally embed PLLs able to output between 20 MHz and 100 MHz and do not embed the divider by 2. The ARM9-based systems generally embed PLLs able to output between 80 MHz and 240 MHz. As the 48 MHz required by the USB cannot be reached by such a PLL, the optional divider by 2 is implemented.

The block diagram of the Clock Generator is shown in Figure 36.

Figure 36. Clock Generator Block Diagram

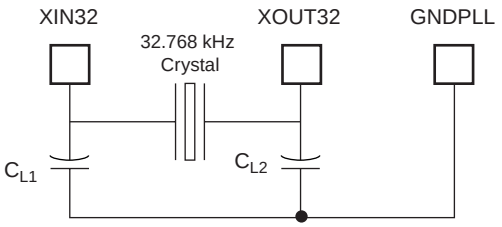


Slow Clock Oscillator

Slow Clock Oscillator Connection

The Clock Generator integrates a low-power 32.768 kHz oscillator. The XIN32 and XOUT32 pins must be connected to a 32.768 kHz crystal. Two external capacitors must be wired as shown in Figure 37.

Figure 37. Typical Slow Clock Oscillator Connection



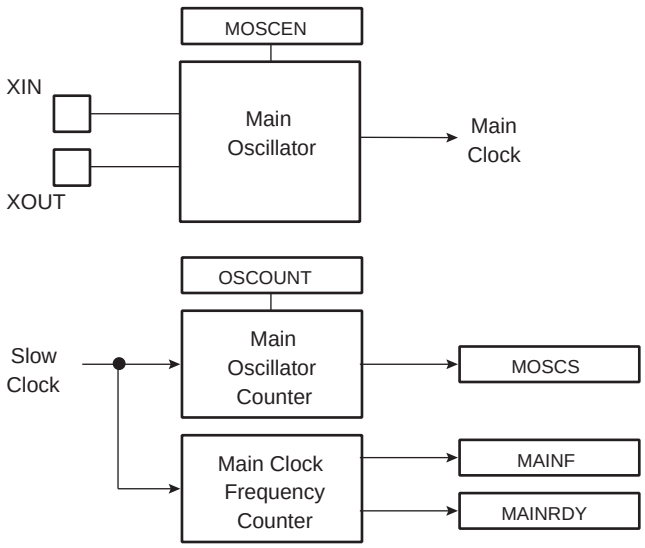
Slow Clock Oscillator Startup Time

The startup time of the Slow Clock Oscillator is given in the section “DC Characteristics” on page 432. As it is often higher than 500 ms and the processor requires an assertion of the reset until it has stabilized, the user must implement an external reset supervisor covering this startup time. However, this startup is only required in case of cold reset, i.e. in case of system power-up. When a warm reset occurs, the length of the reset pulse may be much lower. For further details, see the section “Reset Controller” on page 77.

Main Oscillator

Figure 38 shows the Main Oscillator block diagram.

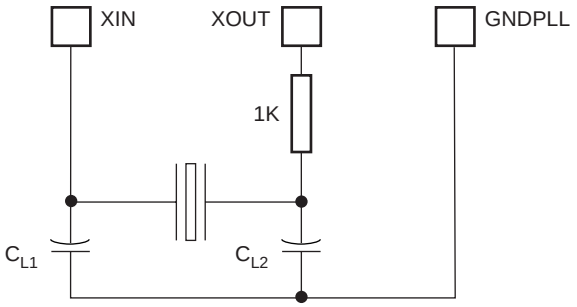
Figure 38. Main Oscillator Block Diagram



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 39. The 1 kΩ resistor is only required for crystals with frequencies lower than 8 MHz. The oscillator contains twenty-five 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 section “DC Characteristics” on page 432.

Figure 39. Typical Crystal Connection



Main Oscillator Startup Time

The startup time of the Main Oscillator is given in the section “DC Characteristics” on page 432. The startup time depends on the crystal frequency and increases 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 the Slow Clock mode 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 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 on this event.

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 bit MAINRDY 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 pin XIN. The input characteristics of the XIN pin under these conditions are given in the product electrical characteristics section. The programmer has to be sure not to modify the MOSCEN bit in the Main Oscillator Register (CKGR_MOR). This bit must remain at 0, its reset value, for the external clock to operate properly. While this bit is at 0, the pin XIN is tied low to prevent any internal oscillation regardless of pin connected.

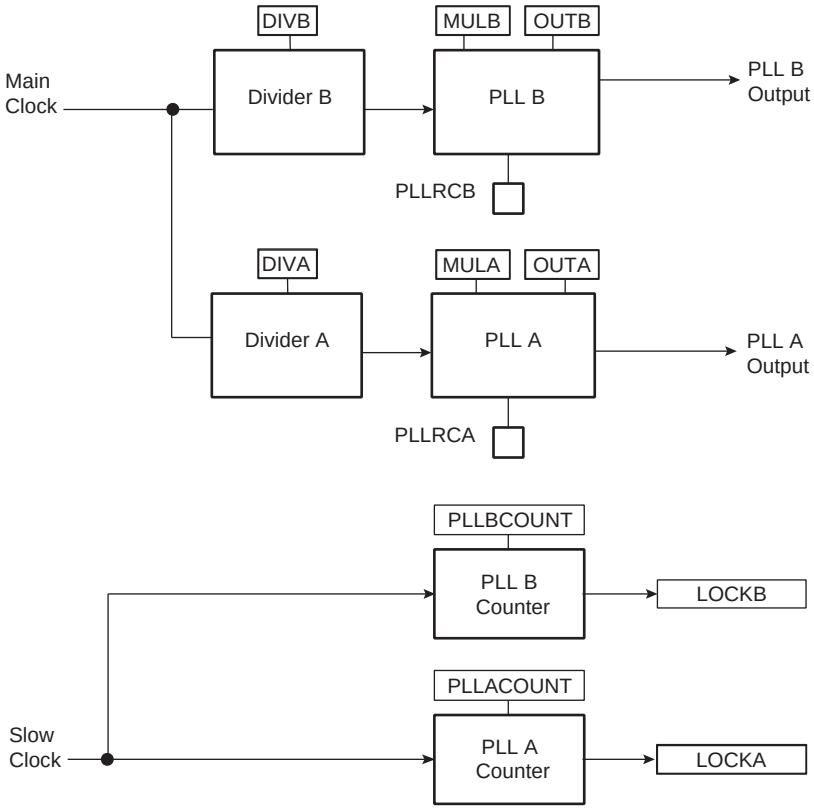
The external clock signal must meet the requirements relating to the power supply VDDPLL (i.e., between 1.65V and 1.95V) and cannot exceed 50 MHz.

Divider and PLL Blocks

The Clock Generator features two Divider/PLL Blocks that generates a wide range of frequencies. Additionally, they provide a 48 MHz signal to the embedded USB device and/or host ports, regardless of the frequency of the Main Clock.

Figure 40 shows the block diagram of the divider and PLL blocks.

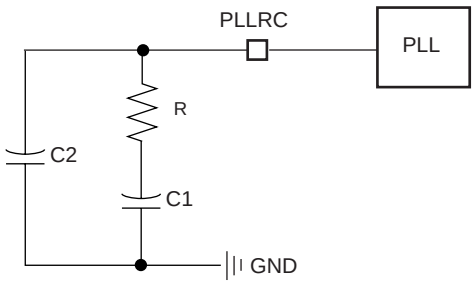
Figure 40. Divider and PLL Blocks Block Diagram



PLL Filters

The two PLLs require connection to an external second-order filter through the pins PLLRC. Figure 41 shows a schematic of these filters.

Figure 41. PLL Capacitors and Resistors



Values of R, C1 and C2 to be connected to the PLLRC pins 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.

PLL Source Clock

The source of PLLs A and B is respectively the output of Divider A, i.e., the Main Clock divided by DIVA, and the output of Divider B, i.e., the Main Clock divided by DIVB.

As the input frequency of the PLLs is limited, the user has to make sure that the programming of DIVA and DIVB are compliant with the input frequency range of the PLLs, which is given in the section “DC Characteristics” on page 432.

Divider and Phase Lock Loop Programming

The two dividers increase the accuracy of the PLLA and the PLLB clocks independently of the input frequency.

The Main Clock can be divided by programming the DIVB field in CKGR_PLLBR and the DIVA field in CKGR_PLLAR. Each divider can be set between 1 and 255 in steps of 1. When the DIVA and DIVB fields are set to 0, the output of the divider and the PLL outputs A and B are a continuous signal at level 0. On reset, the DIVA and DIVB fields are set to 0, thus both PLL input clocks are set to 0.

The two PLLs of the clock generator allow multiplication of the divider’s outputs. The PLLA and the PLLB clock signals have a frequency that depends on the respective source signal frequency and on the parameters DIV (DIVA, DIVB) and MUL (MULA, MULB). The factor applied to the source signal frequency is (MUL + 1)/DIV. When MULA or MULB is written to 0, the corresponding PLL is disabled and its power consumption is saved. Re-enabling the PLLA or the PLLB can be performed by writing a value higher than 0 in the MULA or MULB field, respectively.

Whenever a PLL is re-enabled or one of its parameters is changed, the LOCKA or LOCKB bit in PMC_SR is automatically cleared. The values written in the PLLACOUNT or PLLBCOUNT fields in CKGR_PPLAR and CKGR_PLLBR, respectively, are loaded in the corresponding PLL counter. The PLL counter then decrements at the speed of the Slow Clock until it reaches 0. At this time, the corresponding 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 PLLACOUNT and PLLBCOUNT field. The transient time depends on the PLL filters. The initial state of the PLL and its target frequency can be calculated using a specific tool provided by Atmel.

PLLB Divider by 2

In ARM9-based systems, the PLLB clock may be divided by two. This divider can be enabled by setting the bit USB_96M of CKGR_PLLBR. In this case, the divider by 2 is enabled and the PLLB must be programmed to output 96 MHz and not 48 MHz, thus ensuring correct operation of the USB bus.

Clock Controllers

The Power Management Controller provides the clocks to the different peripherals of the system, either internal or external. It embeds the following elements:

- the Master Clock Controller, that selects the Master Clock.
- the Processor Clock Controller, that implements the Idle Mode.
- the Peripheral Clock Controller, that provides power saving by controlling clocks of the embedded peripherals.
- the USB Clock Controller, that distributes the 48 MHz clock to the USB controllers.
- the Programmable Clock Controller, that allows generation of up to four programmable clock signals on external 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 enables Slow Clock Mode by providing a 32.768 kHz signal to the whole device. Selecting the Main Clock saves power consumption of both PLLs, but

prevents using the USB ports. Selecting the PLLB Clock saves the power consumption of the PLLA by running the processor and the peripheral at 48 MHz required by the USB ports. Selecting the PLLA Clock runs the processor and the peripherals at their maximum speed while running the USB ports at 48 MHz.

The Master Clock Controller is made up of a clock selector and a prescaler, as shown in Figure 42. It also contains an optional Master Clock divider in products integrating an ARM9 processor. This allows the processor clock to be faster than the Master Clock.

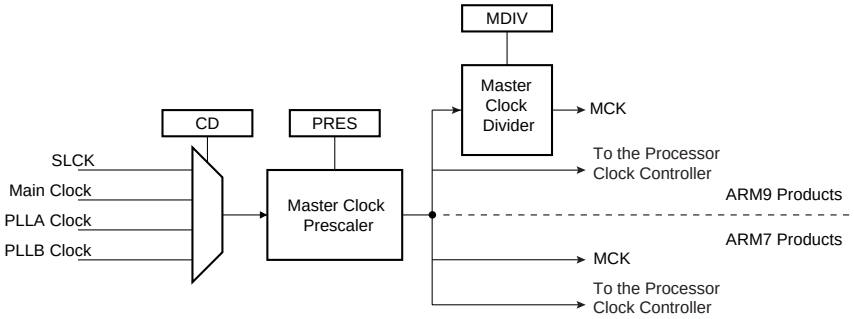
The Master Clock selection is made by writing the CSS field (Clock Source Selection) in PMC_MCKR (Master Clock Register). The prescaler supports the division by a power of 2 of the selected clock between 1 and 64. The PRES field in PMC_MCKR programs the prescaler.

When the Master Clock divider is implemented, it can be programmed between 1 and 4 through the MDIV field in PMC_MCKR.

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.

Note: A new value to be written in PMC_MCKR must not be the same as the current value in PMC_MCKR.

Figure 42. Master Clock Controller



Processor Clock Controller

The PMC features a Processor Clock Controller that implements the 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).

Processor Clock Source

The clock provided to the processor is determined by the Master Clock controller. On ARM7-based systems, the Processor Clock source is directly the Master Clock.

On ARM9-based systems, the Processor Clock source might be 2, 3 or 4 times the Master Clock. This ratio value is determined by programming the field MDIV of the Master Clock Register (PMC_MCKR).

Idle Mode

The Processor Clock is enabled after a reset and is automatically re-enabled by any enabled interrupt. The 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 PMC controls the clocks of each embedded peripheral. 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. When the clock is re-enabled, the peripheral resumes action where it left off. 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.

USB Clock Controller

If using one of the USB ports, the user has to program the Divider and PLL B block to output a 48 MHz signal with an accuracy of ± 0.25%.

When the clock for the USB is stable, the USB device and host clocks, UDPCK and UHPCK, can be enabled. They can be disabled when the USB transactions are finished, so that the power consumption generated by the 48 MHz signal on these peripherals is saved.

The USB ports require both the 48 MHz signal and the Master Clock. The Master Clock may be controlled via the Peripheral Clock Controller.

USB Device Clock Control

The USB Device Port clock UDPCK can be enabled by writing 1 at the UDP bit in PMC_SCER (System Clock Enable Register) and disabled by writing 1 at the bit UDP in PMC_SCDR (System Clock Disable Register). The activity of UDPCK is shown in the bit UDP of PMC_SCSR (System Clock Status Register).

USB Device Port Suspend

When the USB Device Port detects a suspend condition, the 48 MHz clock is automatically disabled, i.e., the UDP bit in PMC_SCSR is cleared. It is also possible to automatically disable the Master Clock provided to the USB Device Port on a suspend condition. The MCKUDP bit in PMC_SCSR configures this feature and can be set or cleared by writing one in the same bit of PMC_SCER and PMC_SCDR.

USB Host Clock Control

The USB Host Port clock UHPCK can be enabled by writing 1 at the UHP bit in PMC_SCER (System Clock Enable Register) and disabled by writing 1 at the UHP bit in PMC_SCDR (System Clock Disable Register). The activity of UDPCK is shown in the bit UHP of PMC_SCSR (System Clock Status Register).

Programmable Clock Output Controller

The PMC controls up to four signals to be output on external pins PCK0 to PCK3. Each signal can be independently programmed via the registers PMC_PCK0 to PMC_PCK3.

PCK0 to PCK3 can be independently selected between the four clocks provided by the Clock Generator by writing the CSS field in PMC_PCK0 to PMC_PCK3. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the field PRES (Prescaler) in PMC_PCK0 to PMC_PCK3.

Each output signal can be enabled and disabled by writing 1 in the corresponding bit PCK0 to PCK3 of PMC_SCER and PMC_SCDR, respectively. Status of the active programmable output clocks are given in the bits PCK0 to PCK3 of PMC_SCSR (System Clock Status Register).

Moreover, like the MCK, 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.

Note also that it is required to assign the pin to the Programmable Clock operation in the PIO Controller to enable the signal to be driven on the pin.

Clock Switching Details

Master Clock Switching Timings

Table 30 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 30. Clock Switching Timings (Worst Case)

From / To	Main Clock	SLCK	PLLA Clock	PLLB Clock
Main Clock	–	4 x SLCK + 2.5 x Main Clock	3 x PLLA Clock + 4 x SLCK + 1 x Main Clock	3 x PLLB Clock + 4 x SLCK + 1 x Main Clock
SLCK	0.5 x Main Clock + 4.5 x SLCK	–	3 x PLLA Clock + 5 x SLCK	3 x PLLB Clock + 5 x SLCK
PLLA Clock	0.5 x Main Clock + 4 x SLCK + PLLACOUNT x SLCK + 2.5 x PLLA Clock	2.5 x PLLA Clock + 5 x SLCK + PLLACOUNT x SLCK	2.5 x PLLA Clock + 4 x SLCK + PLLB COUNT x SLCK	3 x PLLA Clock + 4 x SLCK + 1.5 x PLLA Clock
PLLB Clock	0.5 x Main Clock + 4 x SLCK + PLLBCOUNT x SLCK + 2.5 x PLLB Clock	2.5 x PLLB Clock + 5 x SLCK + PLLBCOUNT x SLCK	3 x PLLB Clock + 4 x SLCK + 1.5 x PLLB Clock	2.5 x PLLB Clock + 4 x SLCK + PLLACOUNT x SLCK

Clock Switching Waveforms

Figure 43. Switch Master Clock from Slow Clock to PLLA Clock

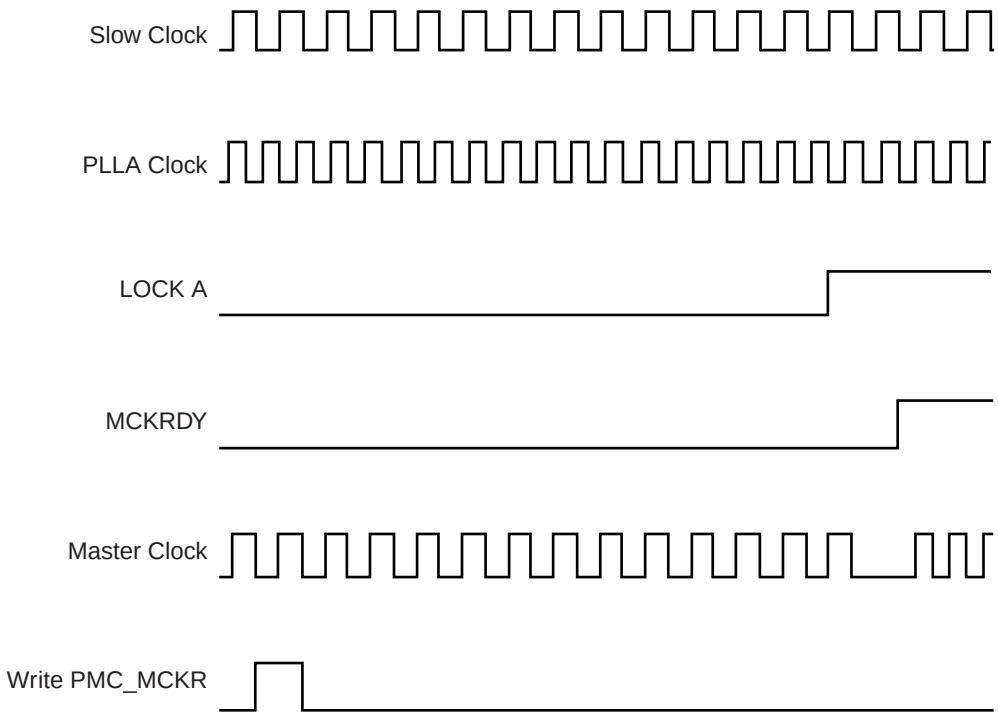


Figure 44. Switch Master Clock from Main Clock to Slow Clock

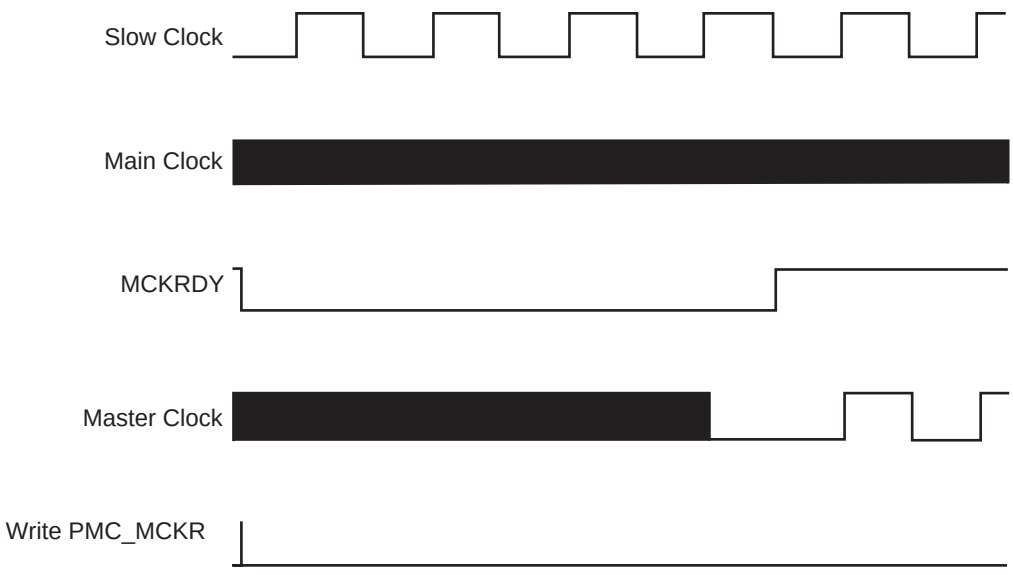


Figure 45. Change PLLA Programming

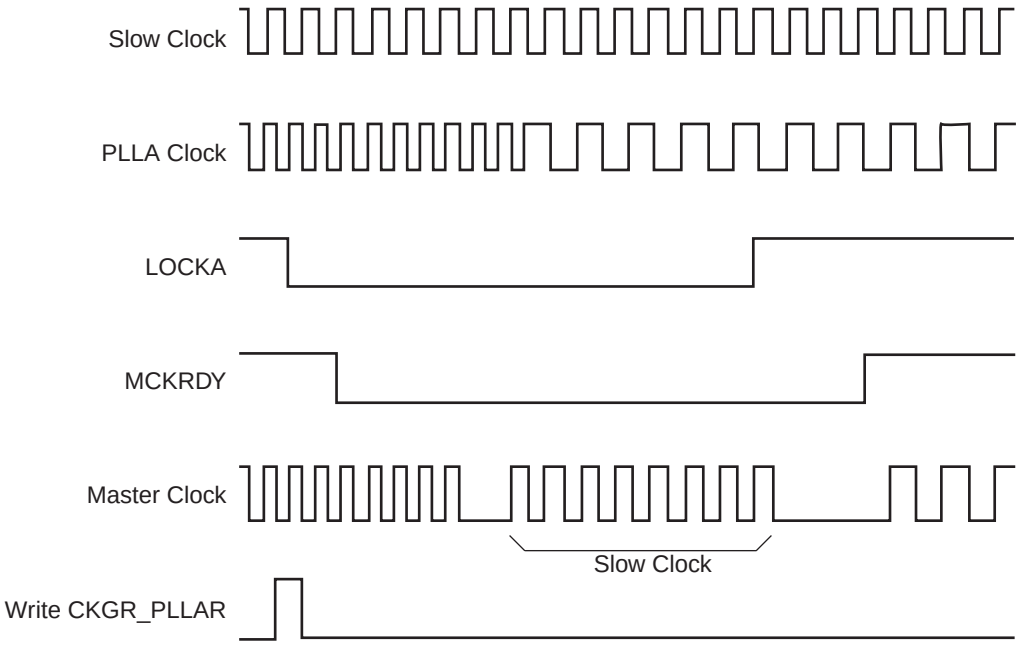
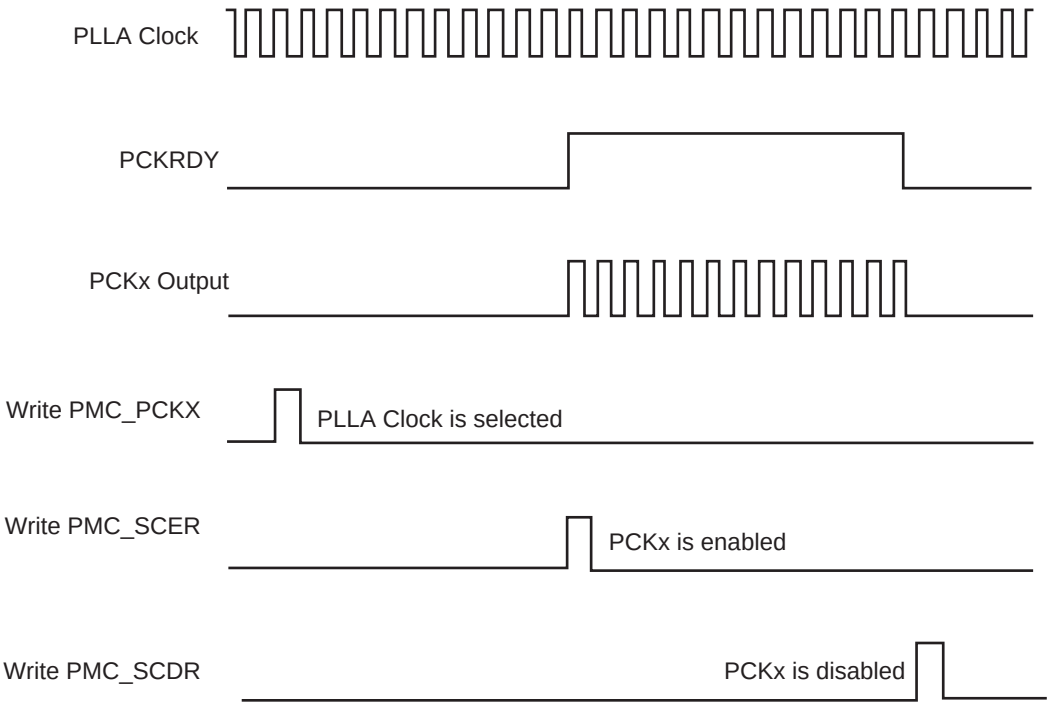


Figure 46. Programmable Clock Output Programming



Power Management Controller (PMC) User Interface

Table 31. 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	ReadWrite	0x0
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read-only	-
0x0028	PLL A Register	CKGR_PLLAR	ReadWrite	0x3F00
0x002C	PLL B Register	CKGR_PLLBR	ReadWrite	0x3F00
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x00
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
0x0048	Programmable Clock 2 Register	PMC_PCK2	Read/Write	0x0
0x004C	Programmable Clock 3 Register	PMC_PCK3	Read/Write	0x0
0x0050	Reserved	–	–	–
0x0054	Reserved	–	–	–
0x0058	Reserved	–	–	–
0x005C	Reserved	–	–	–
0x0060	Interrupt Enable Register	PMC_IER	Write-only	--
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	--
0x0068	Status Register	PMC_SR	Read-only	--
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0

PMC System Clock Enable Register

Register Name: PMC_SCER

Access Type: Write-only

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

- **PCK: Processor Clock Enable**

0 = No effect.

1 = Enables the Processor Clock.

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

0 = No effect.

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

- **MCKUDP: USB Device Port Master Clock Automatic Disable on Suspend Enable**

0 = No effect.

1 = Enables the automatic disable of the Master Clock of the USB Device Port when a suspend condition occurs.

- **UHP: USB Host Port Clock Enable**

0 = No effect.

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

- **PCK0...PCK3: Programmable Clock 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
–	–	–	–	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	UHP	–	MCKUDP	UDP	PCK

- **PCK: Processor Clock Disable**

0 = No effect.

1 = Disables the Processor Clock.

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

0 = No effect.

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

- **MCKUDP: USB Device Port Master Clock Automatic Disable on Suspend Disable**

0 = No effect.

1 = Disables the automatic disable of the Master Clock of the USB Device Port when a suspend condition occurs.

- **UHP: USB Host Port Clock Disable**

0 = No effect.

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

- **PCK0...PCK3: Programmable Clock 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
–	–	–	–	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
–	–	–	UHP	–	MCKUDP	UDP	PCK

- **PCK: Processor Clock Status**

0 = The Processor Clock is disabled.

1 = The Processor Clock is enabled.

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

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

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

- **MCKUDP: USB Device Port Master Clock Automatic Disable on Suspend Status**

0 = The automatic disable of the Master clock of the USB Device Port when suspend condition occurs is disabled.

1 = The automatic disable of the Master clock of the USB Device Port when suspend condition occurs is enabled.

- **UHP: USB Host Port Clock Status**

0 = The 48 MHz clock of the USB Host Port is disabled.

1 = The 48 MHz clock of the USB Host Port is enabled.

- **PCK0...PCK3: Programmable Clock 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	–	–

- **PID2...PID31: Peripheral Clock Enable**

0 = No effect.

1 = Enables the corresponding peripheral clock.

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	–	–

- **PID2...PID31: Peripheral Clock 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	–	–

- **PID2...PID31: Peripheral Clock 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
–	–	–	–	–	–	-	MOSCEN

- **MOSCEN: Main Oscillator Enable**

0 = The Main Oscillator is disabled. Main Clock is the signal connected on XIN.

1 = The Main Oscillator is enabled. A crystal must be connected between XIN and XOUT.

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

Specifies the number of Slow Clock cycles 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 A Register

Register Name: CKGR_PLLAR
Access Type: Read/Write

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

23	22	21	20	19	18	17	16
MULA							

15	14	13	12	11	10	9	8
OUTA		PLLACOUNT					

7	6	5	4	3	2	1	0
DIVA							

Possible limitations on PLLA input frequencies and multiplier factors should be checked before using the Clock Generator.

- **DIVA: Divider A**

DIVA	Divider Selected
0	Divider output is 0
1	Divider is bypassed
2 - 255	Divider output is the Main Clock divided by DIVA.

- **PLLACOUNT: PLL A Counter**

Specifies the number of Slow Clock cycles before the LOCKA bit is set in PMC_SR after CKGR_PLLAR is written.

- **OUTA: PLLA Clock Frequency Range**

OUTA		PLLA Frequency Output Range
0	0	80 MHz to 160 MHz
0	1	Reserved
1	0	150 MHz to 240 MHz
1	1	Reserved

- **MULA: PLL A Multiplier**

0 = The PLL A is deactivated.

1 up to 2047 = The PLLA Clock frequency is the PLLA input frequency multiplied by MULA + 1.

PMC Clock Generator PLL B Register

Register Name: CKGR_PLLBR
Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	–	USB_96M	–	MULB		

23	22	21	20	19	18	17	16
MULB							

15	14	13	12	11	10	9	8
OUTB		PLLBCOUNT					

7	6	5	4	3	2	1	0
DIVB							

- **DIVB: Divider B**

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

- **PLLBCOUNT: PLL B Counter**

Specifies the number of slow clock cycles before the LOCKB bit is set in PMC_SR after CKGR_PLLBR is written.

- **OUTB: PLLB Clock Frequency Range**

OUTB		PLLB Clock Frequency Range
0	0	80 MHz to 160 MHz
0	1	Reserved
1	0	150 MHz to 240 MHz
1	1	Reserved

- **MULB: PLL B Multiplier**

0 = The PLL B is deactivated.

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

- **USB_96M: Divider by 2 Enable (only on ARM9-based Systems)**

0 = USB ports clocks are PLLB Clock, therefore the PMC Clock Generator must be programmed for the PLLB Clock to be 48 MHz.

1 = USB ports clocks are PLLB Clock divided by 2, therefore the PMC Clock Generator must be programmed for the PLLB Clock to be 96 MHz.

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
–	–	–	–	–	–	MDIV	
7	6	5	4	3	2	1	0
–	–		PRES			CSS	

Note: Value to be written in PMC_MCKR must not be the same as current value in PMC_MCKR.

- **CSS: Master Clock Selection**

CSS		Clock Source Selection
0	0	Slow Clock is selected
0	1	Main Clock is selected
1	0	PLL A Clock is selected
1	1	PLL B 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

- **MDIV: Master Clock Division (on ARM9-based systems only)**

0 = The Master Clock and the Processor Clock are the same.

1 = The Processor Clock is twice as fast as the Master Clock.

2 = The Processor Clock is three times faster than the Master Clock.

3 = The Processor Clock is four times faster than the Master Clock.

PMC Programmable Clock Register 0 to 3

Register Name: PMC_PCK0 - PMC-PCK3

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	PLL A Clock is selected
1	1	PLL B 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
–	–	–	–	PCK3RDY	PCK2RDY	PCK1RDY	PCK0RDY
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: Main Oscillator Status**
- **LOCKA: PLL A Lock**
- **LOCKB: PLL B Lock**
- **MCKRDY: Master Clock Ready**
- **PCK0RDY - PCK3RDY: Programmable Clock Ready**

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
–	–	–	–	PCK3RDY	PCK2RDY	PCK1RDY	PCK0RDY
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: Main Oscillator Status**
- **LOCKA: PLL A Lock**
- **LOCKB: PLL B Lock**
- **MCKRDY: Master Clock Ready**
- **PCK0RDY - PCK3RDY: Programmable Clock Ready**

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
–	–	–	–	PCK3RDY	PCK2RDY	PCK1RDY	PCK0RDY
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: MOSCS Flag Status**

0 = Main oscillator is not stabilized.

1 = Main oscillator is stabilized.

- **LOCKA: PLLA Lock Status**

0 = PLLL A is not locked

1 = PLL A is locked.

- **LOCKB: PLLB Lock Status**

0 = PLL B is not locked.

1 = PLL B is locked.

- **MCKRDY: Master Clock Status**

0 = MCK is not ready.

1 = MCK is ready.

- **PCK0RDY - PCK3RDY: Programmable Clock Ready Status**

0 = Programmable Clock 0 to 3 is not ready.

1 = Programmable Clock 0 to 3 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
–	–	–	–	PCK3RDY	PCK2RDY	PCK1RDY	PCK0RDY
7	6	5	4	3	2	1	0
–	–	–	–	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: Main Oscillator Status**
- **LOCKA: PLL A Lock**
- **LOCKB: PLL B Lock**
- **MCKRDY: Master Clock Ready**
- **PCK0RDY - PCK3RDY: Programmable Clock Ready**
- **MOSCS: MOSCS Interrupt Mask**

0 = The corresponding interrupt is enabled.

1 = The corresponding interrupt is disabled.

System Timer (ST)

Overview

The System Timer (ST) module integrates three different free-running timers:

- A Period Interval Timer (PIT) that sets the time base for an operating system.
- A Watchdog Timer (WDT) with system reset capabilities in case of software deadlock.
- A Real-Time Timer (RTT) counting elapsed seconds.

These timers count using the Slow Clock provided by the Power Management Controller. Typically, this clock has a frequency of 32.768 kHz, but the System Timer might be configured to support another frequency.

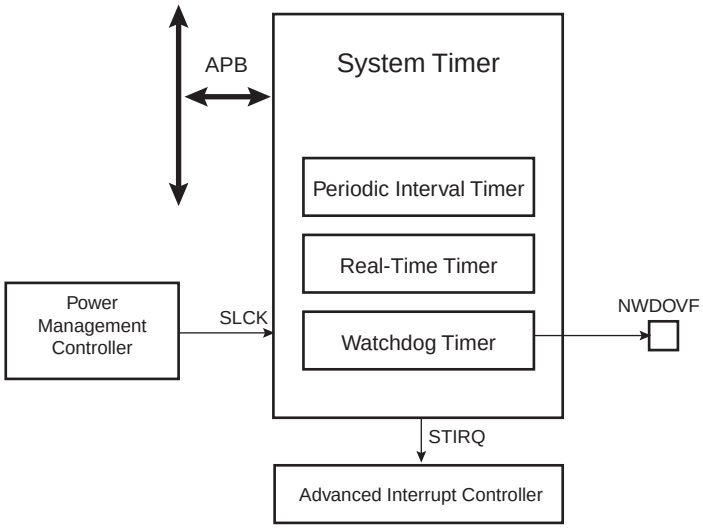
The System Timer provides an interrupt line connected to one of the sources of the Advanced Interrupt Controller (AIC). Interrupt handling requires programming the AIC before configuring the System Timer. Usually, the System Timer interrupt line is connected to the first interrupt source line and shares this entry with the Debug Unit (DBGU) and the Real Time Clock (RTC). This sharing requires the programmer to determine the source of the interrupt when the source 1 is triggered.

Important features of the System Timer include:

- One Period Interval Timer, 16-bit Programmable Counter
- One Watchdog Timer, 16-bit Programmable Counter
- One Real-time Timer, 20-bit Free-running Counter
- Interrupt Generation on Event

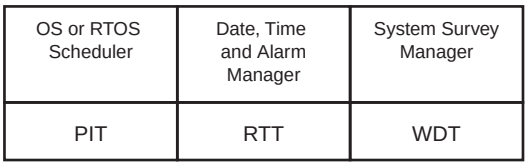
Block Diagram

Figure 47. System Timer Block Diagram



Application Block Diagram

Figure 48. Application Block Diagram



Product Dependencies

Power Management

The System Timer is continuously clocked at 32768 Hz. The power management controller has no effect on the system timer behavior.

Interrupt Sources

The System Timer interrupt is generally connected to the source 1 of the Advanced Interrupt Controller. This interrupt line is the result of the OR-wiring of the system peripheral interrupt lines (System Timer, Real Time Clock, Power Management Controller, Memory Controller). When a system interrupt happens, the service routine must first determine the cause of the interrupt. This is accomplished by reading successively the status registers of the above mentioned system peripherals.

Watchdog Overflow

The System Timer is capable of driving the NWDOVF pin. This pin might be implemented or not in a product. When it is implemented, this pin might or not be multiplexed on the PIO Controllers even though it is recommended to dedicate a pin to the watchdog function. If the NWDOVF is multiplexed on a PIO Controller, this last should be first programmed to assign the pin to the watchdog function before using the pin as NWDOVF.

When it is not implemented, programming the associated bits and registers has no effect on the behavior of the System Timer.

Functional Description

System Timer Clock

The System Timer uses only the SLCK clock so that it is capable to provide periodic, watchdog, second change or alarm interrupt even if the Power Management Controller is programmed to put the product in Slow Clock Mode. If the product has the capability to back up the Slow Clock oscillator and the System Timer, the System Timer can continue to operate.

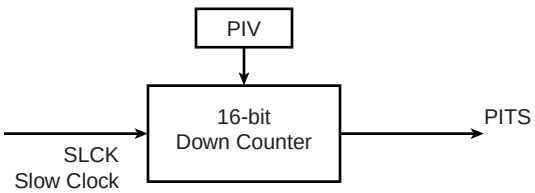
Period Interval Timer (PIT)

The Period Interval Timer can be used to provide periodic interrupts for use by operating systems. The reset value of the PIT is 0 corresponding to the maximum value. It is built around a 16-bit down counter, which is preloaded by a value programmed in ST_PIMR (Period Interval Mode Register). When the PIT counter reaches 0, the bit PITS is set in ST_SR (Status Register), and an interrupt is generated if it is enabled.

The counter is then automatically reloaded and restarted. Writing to the ST_PIMR at any time immediately reloads and restarts the down counter with the new programmed value.

Warning: If ST_PIMR is programmed with a period less or equal to the current MCK period, the update of the PITS status bit and its associated interrupt generation are unpredictable.

Figure 49. Period Interval Timer



Watchdog Timer (WDT)

The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It is built around a 16-bit down counter loaded with the value defined in ST_WDMR (Watchdog Mode Register).

At reset, the value of the ST_WDMR is 0x00020000, corresponding to the maximum value of the counter. The watchdog overflow signal is tied low during 8 slow clock cycles when a watchdog overflow occurs (EXTEN bit set in ST_WDMR).

It uses the Slow Clock divided by 128 to establish the maximum watchdog period to be 256 seconds (with a typical slow clock of 32.768 kHz).

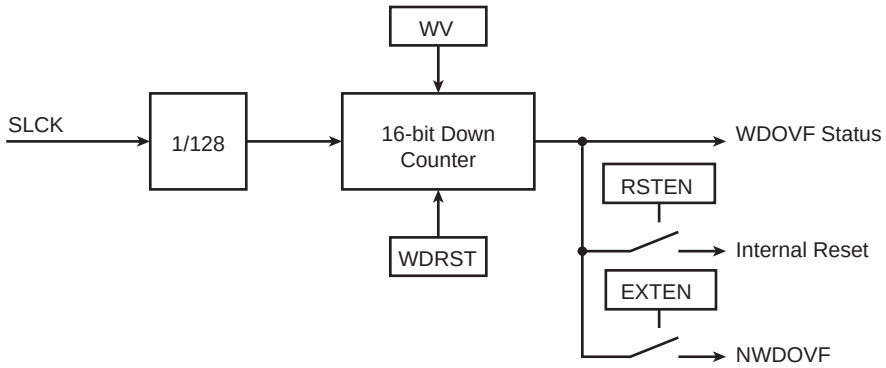
In normal operation, the user reloads the Watchdog at regular intervals before the timer overflow occurs, by setting the bit WDRST in the ST_CR (Control Register).

If an overflow does occur, the watchdog timer:

- Sets the WDOVF bit in ST_SR (Status Register), from which an interrupt can be generated.
- Generates a pulse for 8 slow clock cycles on the external signal watchdog overflow if the bit EXTEN in ST_WDMR is set.
- Generates an internal reset if the parameter RSTEN in ST_WDMR is set.
- Reloads and restarts the down counter.

Writing the ST_WDMR does not reload or restart the down counter. When the ST_CR is written the watchdog counter is immediately reloaded from ST_WDMR and restarted and the Slow Clock 128 divider is also immediately reset and restarted.

Figure 50. Watchdog Timer



Real-time Timer (RTT)

The Real-Time Timer is used to count elapsed seconds. It is built around a 20-bit counter fed by Slow Clock divided by a programmable value. At reset, this value is set to 0x8000, corresponding to feeding the real-time counter with a 1 Hz signal when the Slow Clock is 32.768 Hz. The 20-bit counter can count up to 1048576 seconds, corresponding to more than 12 days, then roll over to 0.

The Real-Time Timer value can be read at any time in the register ST_CRTR (Current Real-time Register). As this value can be updated asynchronously to the master clock, it is advisable to read this register twice at the same value to improve accuracy of the returned value.

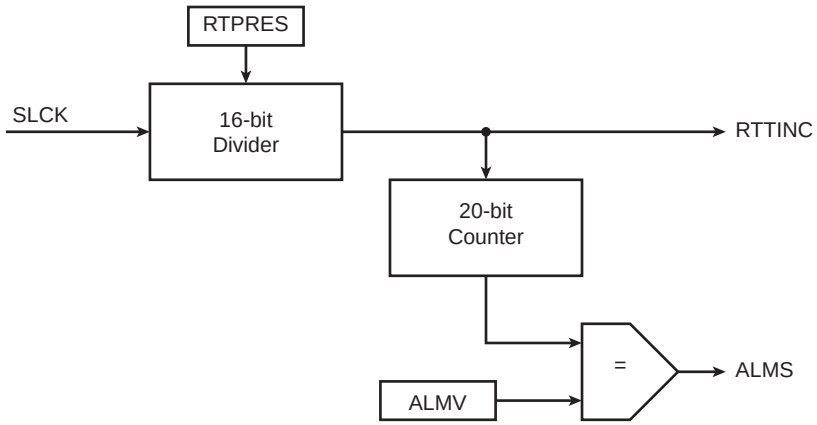
This current value of the counter is compared with the value written in the alarm register ST_RTAR (Real-time Alarm Register). If the counter value matches the alarm, the bit ALMS in TC_SR is set. The alarm register is set to its maximum value, corresponding to 0, after a reset.

The bit RTTINC in ST_SR is set each time the 20-bit counter is incremented. This bit can be used to start an interrupt, or generate a one-second signal.

Writing the ST_RTMR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 20-bit counter.

Warning: If RTPRES is programmed with a period less or equal to the current MCK period, the update of the RTTINC and ALMS status bits and their associated interrupt generation are unpredictable.

Figure 51. Real Time Timer



System Timer (ST) User Interface

Table 32. System Timer Registers

Offset	Register	Name	Access	Reset Value
0x0000	Control Register	ST_CR	Write-only	–
0x0004	Period Interval Mode Register	ST_PIMR	Read/Write	0x00000000
0x0008	Watchdog Mode Register	ST_WDMR	Read/Write	0x00020000
0x000C	Real-time Mode Register	ST_RTMR	Read/Write	0x00008000
0x0010	Status Register	ST_SR	Read-only	–
0x0014	Interrupt Enable Register	ST_IER	Write-only	–
0x0018	Interrupt Disable Register	ST_IDR	Write-only	–
0x001C	Interrupt Mask Register	ST_IMR	Write-only	0x0
0x0020	Real-time Alarm Register	ST_RTAR	Read/Write	0x0
0x0024	Current Real-time Register	ST_CRTR	Read-only	0x0

ST Control Register

Register Name: ST_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
–	–	–	–	–	–	–	WDRST

- **WDRST: Watchdog Timer Restart**

0 = No effect.

1 = Reload the start-up value in the watchdog timer.

ST Period Interval Mode Register

Register Name: ST_PIMR

Access Type: Read/Write

–	–	–	–	–	–	–	–

–	–	–	–	–	–	–	–

PIV

PIV

- **PIV: Period Interval Value**

Defines the value loaded in the 16-bit counter of the period interval timer. The maximum period is obtained by programming PIV at 0x0 corresponding to 65536 slow clock cycles.

ST Watchdog Mode Register

Register Name: ST_WDMR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
WDV							

7	6	5	4	3	2	1	0
WDV							

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 16-bit counter. The maximum period is obtained by programming WDV to 0x0 corresponding to 65536 x 128 slow clock cycles.

- **RSTEN: Reset Enable**

0 = No reset is generated when a watchdog overflow occurs.

1 = An internal reset is generated when a watchdog overflow occurs.

- **EXTEN: External Signal Assertion Enable**

0 = The watchdog_overflow is not tied low when a watchdog overflow occurs.

1 = The watchdog_overflow is tied low during 8 slow clock cycles when a watchdog overflow occurs.

ST Real-Time Mode Register

Register Name: ST_RTMR

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
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. The maximum period is obtained by programming RTPRES to 0x0 corresponding to 65536 slow clock cycles.

ST Status Register

Register Name: ST_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
–	–	–	–	ALMS	RTTINC	WDOVF	PITS

- **PITS: Period Interval Timer Status**

0 = The period interval timer has not reached 0 since the last read of the Status Register.

1 = The period interval timer has reached 0 since the last read of the Status Register.

- **WDOVF: Watchdog Overflow**

0 = The watchdog timer has not reached 0 since the last read of the Status Register.

1 = The watchdog timer has reached 0 since the last read of the Status Register.

- **RTTINC: Real-time Timer Increment**

0 = The real-time timer has not been incremented since the last read of the Status Register.

1 = The real-time timer has been incremented since the last read of the Status Register.

- **ALMS: Alarm Status**

0 = No alarm compare has been detected since the last read of the Status Register.

1 = Alarm compare has been detected since the last read of the Status Register.

ST Interrupt Enable Register

Register Name: ST_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
–	–	–	–	ALMS	RTTINC	WDOVF	PITS

- **PITS: Period Interval Timer Status Interrupt Enable**
- **WDOVF: Watchdog Overflow Interrupt Enable**
- **RTTINC: Real-time Timer Increment Interrupt Enable**
- **ALMS: Alarm Status Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

ST Interrupt Disable Register

Register Name: ST_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
–	–	–	–	ALMS	RTTINC	WDOVF	PITS

- **PITS: Period Interval Timer Status Interrupt Disable**
- **WDOVF: Watchdog Overflow Interrupt Disable**
- **RTTINC: Real-time Timer Increment Interrupt Disable**
- **ALMS: Alarm Status Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

ST Interrupt Mask Register

Register Name: ST_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
–	–	–	–	ALMS	RTTINC	WDOVF	PITS

- **PITS: Period Interval Timer Status Interrupt Mask**
- **WDOVF: Watchdog Overflow Interrupt Mask**
- **RTTINC: Real-time Timer Increment Interrupt Mask**
- **ALMS: Alarm Status Interrupt Mask**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

ST Real-time Alarm Register

Register Name: ST_RTAR

Access Type: Read/Write

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

- **ALMV: Alarm Value**

Defines the alarm value compared with the real-time timer. The maximum delay before ALMS status bit activation is obtained by programming ALMV to 0x0 corresponding to 1048576 seconds.

ST Current Real-Time Register

Register Name: ST_CRTR

Access Type: Read-only

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

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 Clock (RTC)

Overview

The Real-time Clock (RTC) peripheral is designed for very low power consumption.

It combines a complete time-of-day clock with alarm and a two-hundred-year Gregorian calendar, complemented by a programmable periodic interrupt. The alarm and calendar registers are accessed by a 32-bit data bus.

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

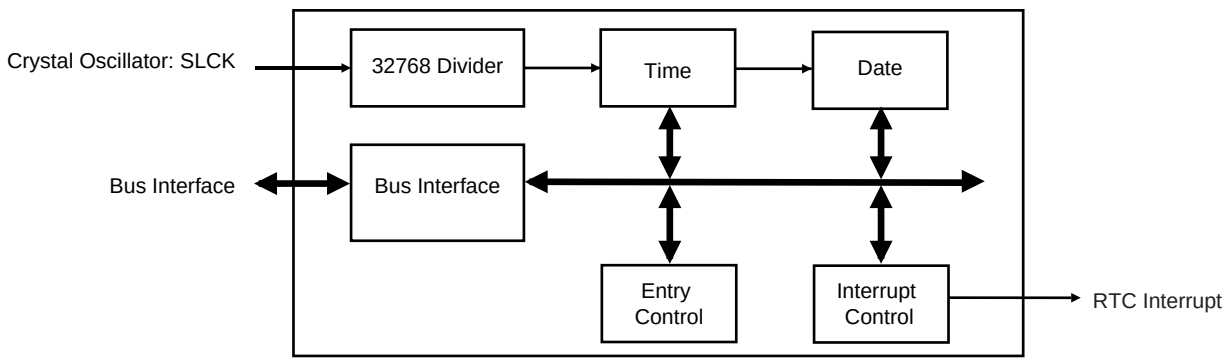
Updating time and calendar fields and configuring the alarm fields are performed by a parallel capture on the 32-bit data bus. An entry control is performed to avoid loading registers with incompatible BCD format data or with an incompatible date according to the current month/year/century.

Important features of the RTC include:

- Low Power Consumption
- Full Asynchronous Design
- Two Hundred Year Calendar
- Programmable Periodic Interrupt
- Alarm and Update Parallel Load
- Control of Alarm and Update Time/Calendar Data In

Block Diagram

Figure 52. RTC Block Diagram



Product Dependencies

Power Management

The Real-time Clock is continuously clocked at 32768 Hz. The Power Management Controller has no effect on RTC behavior.

Interrupt

The RTC Interrupt is connected to interrupt source 1 (IRQ1) of the advanced interrupt controller. This interrupt line is due to the OR-wiring of the system peripheral interrupt lines (System Timer, Real Time Clock, Power Management Controller, Memory Controller, etc.). When a

system interrupt occurs, the service routine must first determine the cause of the interrupt. This is done by reading the status registers of the above system peripherals successively.

Functional Description

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

The valid year range is 1900 to 2099, a two-hundred-year Gregorian calendar achieving full Y2K compliance.

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

Corrections for leap years are included (all years divisible by 4 being leap years, including year 2000). This is correct up to the year 2099.

After hardware reset, the calendar is initialized to Thursday, January 1, 1998.

Reference Clock

The reference clock is Slow Clock (SLCK). It can be driven by the Atmel cell OSC55 or OSC56 (or an equivalent cell) and an external 32.768 kHz crystal.

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

Timing

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

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

Alarm

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

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

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

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

Error Checking

Verification on user interface data is performed when accessing the century, year, month, date, day, hours, minutes, seconds and alarms. A check is performed on illegal BCD entries such as illegal date of the month with regard to the year and century configured.

If one of the time fields is not correct, the data is not loaded into the register/counter and a flag is set in the validity register. The user can not reset this flag. It is reset as soon as an acceptable value is programmed. This avoids any further side effects in the hardware. The same procedure is done for the alarm.

The following checks are performed:

1. Century (check if it is in range 19 - 20)
2. Year (BCD entry check)
3. Date (check range 01 - 31)

4. Month (check if it is in BCD range 01 - 12, check validity regarding “date”)
5. Day (check range 1 - 7)
6. Hour (BCD checks: in 24-hour mode, check range 00 - 23 and check that AM/PM flag is not set if RTC is set in 24-hour mode; in 12-hour mode check range 01 - 12)
7. Minute (check BCD and range 00 - 59)
8. Second (check BCD and range 00 - 59)

Note: If the 12-hour mode is selected by means of the RTC_MODE register, a 12-hour value can be programmed and the returned value on RTC_TIME will be the corresponding 24-hour value. The entry control checks the value of the AM/PM indicator (bit 22 of RTC_TIME register) to determine the range to be checked.

Updating Time/Calendar

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

Then the user must poll or wait for the interrupt (if enabled) of bit ACKUPD in the Status Register. Once the bit reads 1, the user can write to the appropriate register.

Once the update is finished, the user must reset (0) UPDTIM and/or UPDCAL in the Control Register.

When programming the calendar fields, the time fields remain enabled. This avoids a time slip in case the user stays in the calendar update phase for several tens of seconds or more. In successive update operations, the user must wait at least one second after resetting the UPDTIM/UPDCAL bit in the RTC_CR (Control Register) before setting these bits again. This is done by waiting for the SEC flag in the Status Register before setting UPDTIM/UPDCAL bit. After resetting UPDTIM/UPDCAL, the SEC flag must also be cleared.

Real Time Clock (RTC) User Interface

Table 33. RTC Register Mapping

Offset	Register	Register Name	Read/Write	Reset
0x00	RTC Control Register	RTC_CR	Read/Write	0x0
0x04	RTC Mode Register	RTC_MR	Read/Write	0x0
0x08	RTC Time Register	RTC_TIMR	Read/Write	0x0
0x0C	RTC Calendar Register	RTC_CALR	Read/Write	0x01819819
0x10	RTC Time Alarm Register	RTC_TIMALR	Read/Write	0x0
0x14	RTC Calendar Alarm Register	RTC_CALALR	Read/Write	0x01010000
0x18	RTC Status Register	RTC_SR	Read only	0x0
0x1C	RTC Status Clear Command Register	RTC_SCCR	Write only	---
0x20	RTC Interrupt Enable Register	RTC_IER	Write only	---
0x24	RTC Interrupt Disable Register	RTC_IDR	Write only	---
0x28	RTC Interrupt Mask Register	RTC_IMR	Read only	0x0
0x2C	RTC Valid Entry Register	RTC_VER	Read only	0x0

RTC Control Register

Name: RTC_CR

Access Type: Read/Write

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

- **UPDTIM: Update Request Time Register**

0 = No effect.

1 = Stops the RTC time counting.

Time counting consists of second, minute and hour counters. Time counters can be programmed once this bit is set and acknowledged by the bit ACKUPD of the Status Register.

- **UPDCAL: Update Request Calendar Register**

0 = No effect.

1 = Stops the RTC calendar counting.

Calendar counting consists of day, date, month, year and century counters. Calendar counters can be programmed once this bit is set.

- **TIMEVSEL: Time Event Selection**

The event that generates the flag TIMEV in RTC_SR (Status Register) depends on the value of TIMEVSEL.

0 = Minute change.

1 = Hour change.

2 = Every day at midnight.

3 = Every day at noon.

- **CALEVSEL: Calendar Event Selection**

The event that generates the flag CALEV in RTC_SR depends on the value of CALEVSEL.

0 = Week change (every Monday at time 00:00:00).

1 = Month change (every 01 of each month at time 00:00:00).

2, 3 = Year change (every January 1 at time 00:00:00).

RTC Mode Register

Name: RTC_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
–	–	–	–	–	–	–	HRMOD

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

0 = 24-hour mode is selected.

1 = 12-hour mode is selected.

All non-significant bits read zero.

RTC Time Register

Name: RTC_TIMR

Access Type: Read/Write

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

- **SEC: Current Second**

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

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

- **MIN: Current Minute**

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

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

- **HOUR: Current Hour**

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

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

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

0 = AM.

1 = PM.

All non-significant bits read zero.

RTC Calendar Register

Name: RTC_CALR

Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	DATE					

23	22	21	20	19	18	17	16
DAY			MONTH				

15	14	13	12	11	10	9	8
YEAR							

7	6	5	4	3	2	1	0
–	CENT						

- **CENT: Current Century**

The range that can be set is 19 - 20 (BCD).

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

- **YEAR: Current Year**

The range that can be set is 00 - 99 (BCD).

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

- **MONTH: Current Month**

The range that can be set is 01 - 12 (BCD).

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

- **DAY: Current Day**

The range that can be set is 1 - 7 (BCD).

The coding of the number (which number represents which day) is user-defined as it has no effect on the date counter.

- **DATE: Current Date**

The range that can be set is 01 - 31 (BCD).

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

All non-significant bits read zero.

RTC Time Alarm Register

Name: RTC_TIMALR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
HOUREN	AMPM	HOUR					

15	14	13	12	11	10	9	8
MINEN	MIN						

7	6	5	4	3	2	1	0
SECEN	SEC						

- **SEC: Second Alarm**

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

- **SECEN: Second Alarm Enable**

0 = The second-matching alarm is disabled.

1 = The second-matching alarm is enabled.

- **MIN: Minute Alarm**

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

- **MINEN: Minute Alarm Enable**

0 = The minute-matching alarm is disabled.

1 = The minute-matching alarm is enabled.

- **HOUR: Hour Alarm**

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

- **AMPM: AM/PM Indicator**

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

- **HOUREN: Hour Alarm Enable**

0 = The hour-matching alarm is disabled.

1 = The hour-matching alarm is enabled.

RTC Calendar Alarm Register

Name: RTC_CALALR

Access Type: Read/Write

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

- **MONTH: Month Alarm**

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

- **MTHEN: Month Alarm Enable**

0 = The month-matching alarm is disabled.

1 = The month-matching alarm is enabled.

- **DATE: Date Alarm**

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

- **DATEEN: Date Alarm Enable**

0 = The date-matching alarm is disabled.

1 = The date-matching alarm is enabled.

RTC Status Register

Name: RTC_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
–	–	–	CALEV	TIMEV	SEC	ALARM	ACKUPD

- **ACKUPD: Acknowledge for Update**

0 = Time and calendar registers cannot be updated.

1 = Time and calendar registers can be updated.

- **ALARM: Alarm Flag**

0 = No alarm matching condition occurred.

1 = An alarm matching condition has occurred.

- **SEC: Second Event**

0 = No second event has occurred since the last clear.

1 = At least one second event has occurred since the last clear.

- **TIMEV: Time Event**

0 = No time event has occurred since the last clear.

1 = At least one time event has occurred since the last clear.

The time event is selected in the TIMEVSEL field in RTC_CTRL (Control Register) and can be any one of the following events: minute change, hour change, noon, midnight (day change).

- **CALEV: Calendar Event**

0 = No calendar event has occurred since the last clear.

1 = At least one calendar event has occurred since the last clear.

The calendar event is selected in the CALEVSEL field in RTC_CR and can be any one of the following events: week change, month change and year change.

RTC Status Clear Command Register

Name: RTC_SCCR

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
–	–	–	CALCLR	TIMCLR	SECCLR	ALRCLR	ACKCLR

- **Status Flag Clear**

0 = No effect.

1 = Clears corresponding status flag in the Status Register (RTC_SR).

RTC Interrupt Enable Register

Name: RTC_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
–	–	–	CALEN	TIMEN	SECEN	ALREN	ACKEN

- **ACKEN: Acknowledge Update Interrupt Enable**

0 = No effect.

1 = The acknowledge for update interrupt is enabled.

- **ALREN: Alarm Interrupt Enable**

0 = No effect.

1 = The alarm interrupt is enabled.

- **SECEN: Second Event Interrupt Enable**

0 = No effect.

1 = The second periodic interrupt is enabled.

- **TIMEN: Time Event Interrupt Enable**

0 = No effect.

1 = The selected time event interrupt is enabled.

- **CALEN: Calendar Event Interrupt Enable**

0 = No effect.

- 1 = The selected calendar event interrupt is enabled.

RTC Interrupt Disable Register

Name: RTC_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
–	–	–	CALDIS	TIMDIS	SECDIS	ALRDIS	ACKDIS

- **ACKDIS: Acknowledge Update Interrupt Disable**

0 = No effect.

1 = The acknowledge for update interrupt is disabled.

- **ALRDIS: Alarm Interrupt Disable**

0 = No effect.

1 = The alarm interrupt is disabled.

- **SECDIS: Second Event Interrupt Disable**

0 = No effect.

1 = The second periodic interrupt is disabled.

- **TIMDIS: Time Event Interrupt Disable**

0 = No effect.

1 = The selected time event interrupt is disabled.

- **CALDIS: Calendar Event Interrupt Disable**

0 = No effect.

1 = The selected calendar event interrupt is disabled.

RTC Interrupt Mask Register

Name: RTC_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
–	–	–	CAL	TIM	SEC	ALR	ACK

- **ACK: Acknowledge Update Interrupt Mask**

0 = The acknowledge for update interrupt is disabled.

1 = The acknowledge for update interrupt is enabled.

- **ALR: Alarm Interrupt Mask**

0 = The alarm interrupt is disabled.

1 = The alarm interrupt is enabled.

- **SEC: Second Event Interrupt Mask**

0 = The second periodic interrupt is disabled.

1 = The second periodic interrupt is enabled.

- **TIM: Time Event Interrupt Mask**

0 = The selected time event interrupt is disabled.

1 = The selected time event interrupt is enabled.

- **CAL: Calendar Event Interrupt Mask**

0 = The selected calendar event interrupt is disabled.

1 = The selected calendar event interrupt is enabled.

RTC Valid Entry Register

Name: RTC_VER

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
–	–	–	–	NVCALAR	NVTIMALR	NVCAL	NVTIM

- **NVTIM: Non valid Time**

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

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

- **NVCAL: Non valid Calendar**

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

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

- **NVTIMALR: Non valid Time Alarm**

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

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

- **NVCALALR: Non valid Calendar Alarm**

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

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

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.

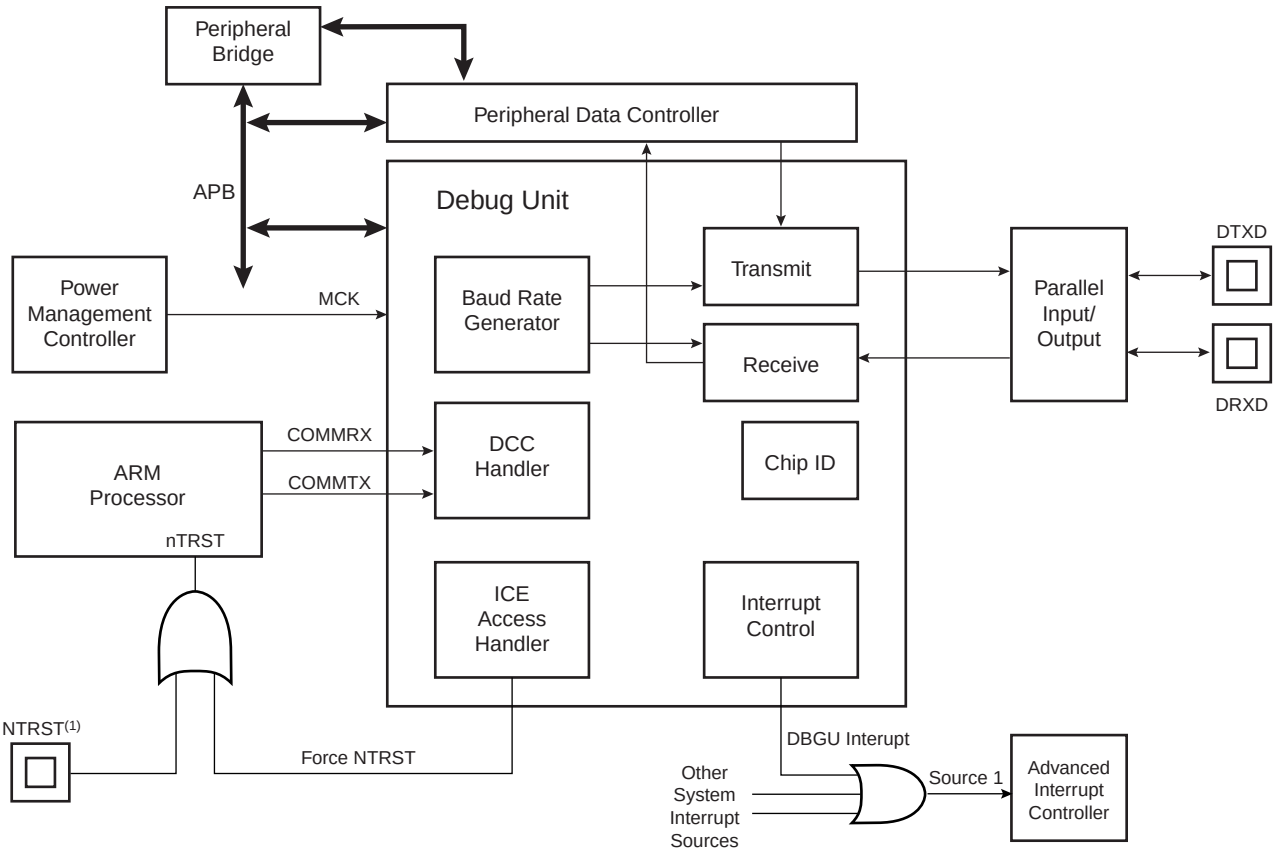
Finally, the Debug Unit features a Force NTRST capability that enables the software to decide whether to prevent access to the system via the In-circuit Emulator. This permits protection of the code, stored in ROM.

Important features of the Debug Unit are:

- System Peripheral to Facilitate Debug of Atmel's ARM-based Systems
- Composed of Four Functions
  - Two-pin UART
  - Debug Communication Channel (DCC) Support
  - Chip ID Registers
  - ICE Access Prevention
- Two-pin UART
  - Implemented Features are 100% Compatible with the Standard Atmel USART
  - Independent Receiver and Transmitter with a Common Programmable Baud Rate Generator
  - Even, Odd, Mark or Space Parity Generation
  - Parity, Framing and Overrun Error Detection
  - Automatic Echo, Local Loopback and Remote Loopback Channel Modes
  - Interrupt Generation
  - Support for Two PDC Channels with Connection to Receiver and Transmitter
- Debug Communication Channel Support
  - Offers Visibility of COMMRX and COMMTX Signals from the ARM Processor
  - Interrupt Generation
- Chip ID Registers
  - Identification of the Device Revision, Sizes of the Embedded Memories, Set of Peripherals
- ICE Access Prevention
  - Enables Software to Prevent System Access Through the ARM Processor's ICE
  - Prevention is Made by Asserting the NTRST Line of the ARM Processor's ICE

Block Diagram

Figure 53. Debug Unit Functional Block Diagram

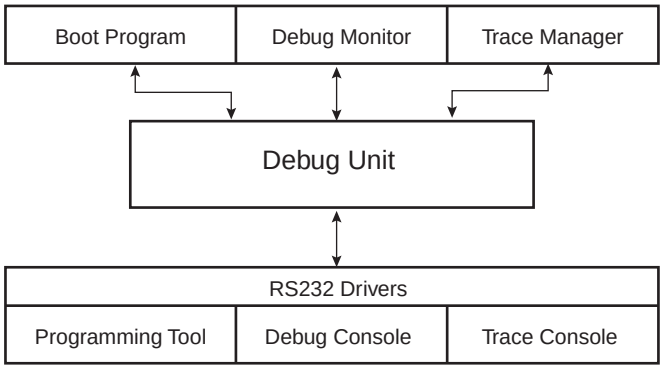


Note: 1. If NTRST pad is not bonded out, it is connected to NRST.

Table 34. Debug Unit Pin Description

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

Figure 54. 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 53. 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.

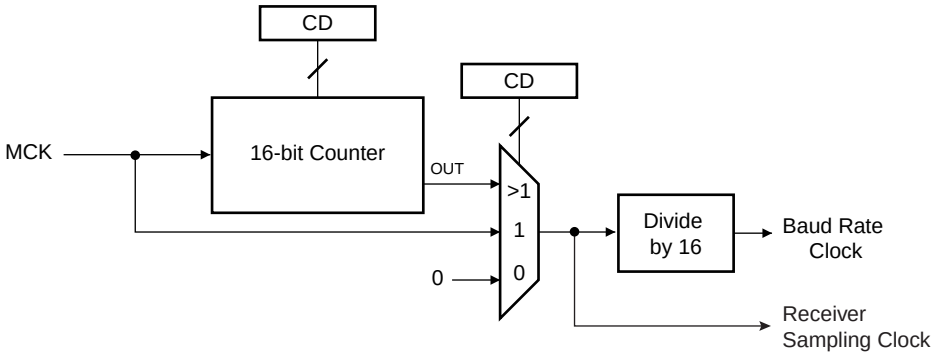
Baud Rate Generator

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 55. 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 56. Start Bit Detection

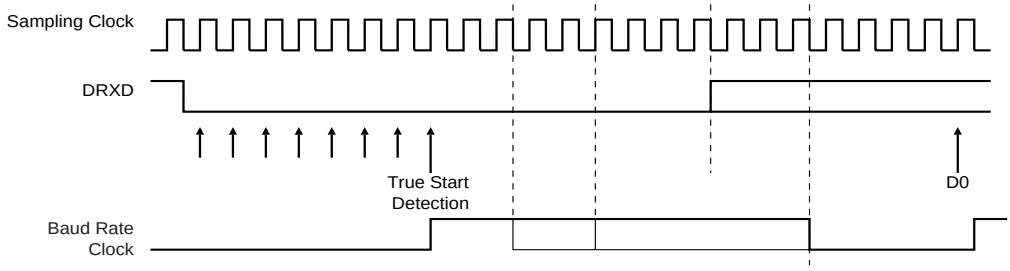
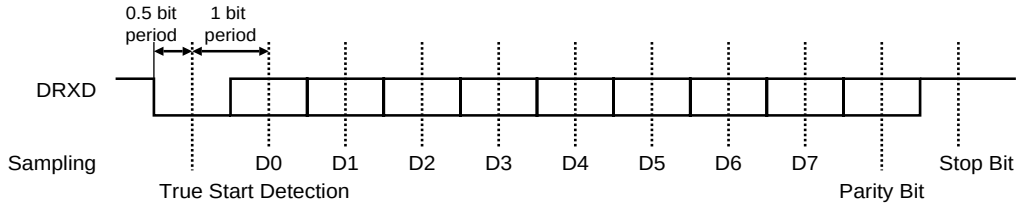


Figure 57. Character Reception

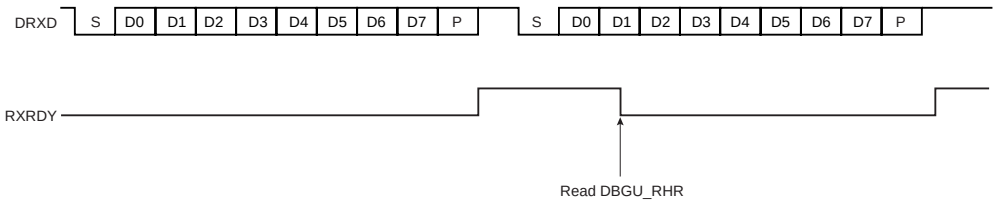
Example: 8-bit, parity enabled 1 stop



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.

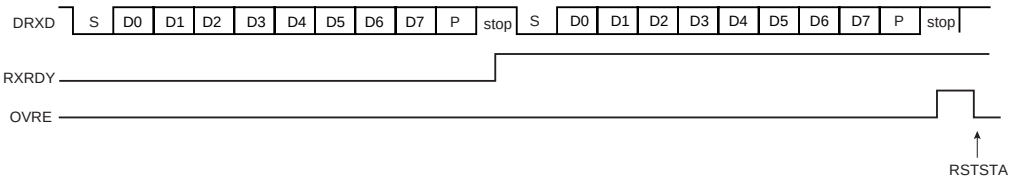
Figure 58. Receiver Ready



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.

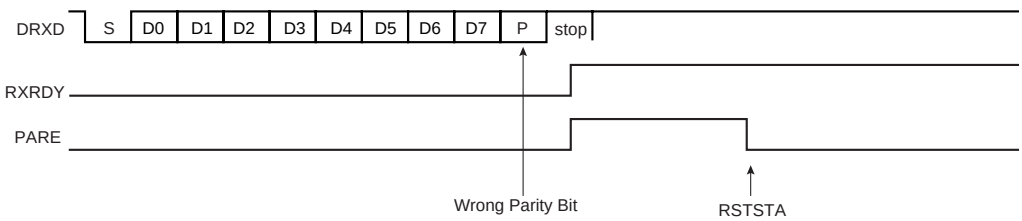
Figure 59. Receiver Overrun



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.

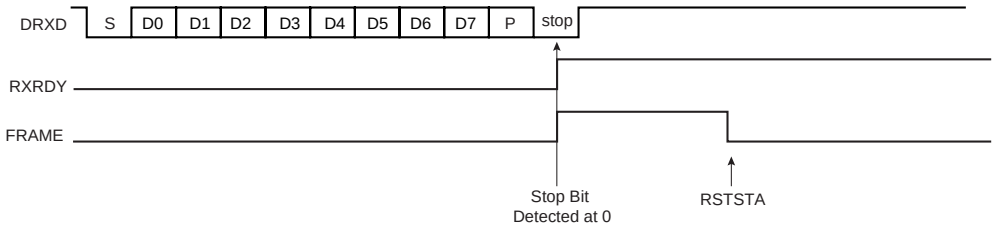
Figure 60. Parity Error



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 61. 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.

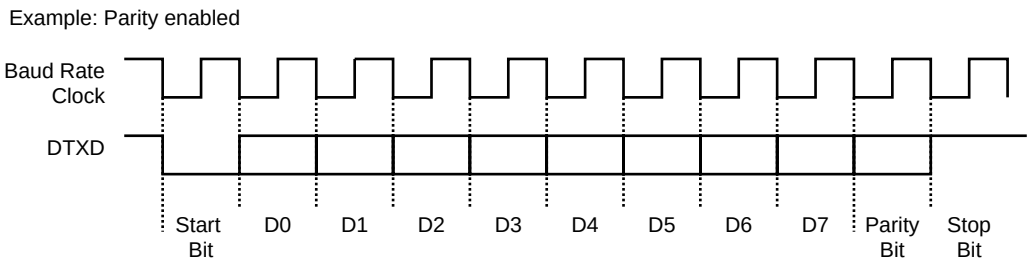
The programmer can disable the transmitter by writing DBGU_CR with the bit TXDIS at 1. If the transmitter is not operating, it is immediately stopped. However, if a character is being processed into the Shift Register and/or a character has been written in the Transmit Holding Register, the characters 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 62. Character Transmission



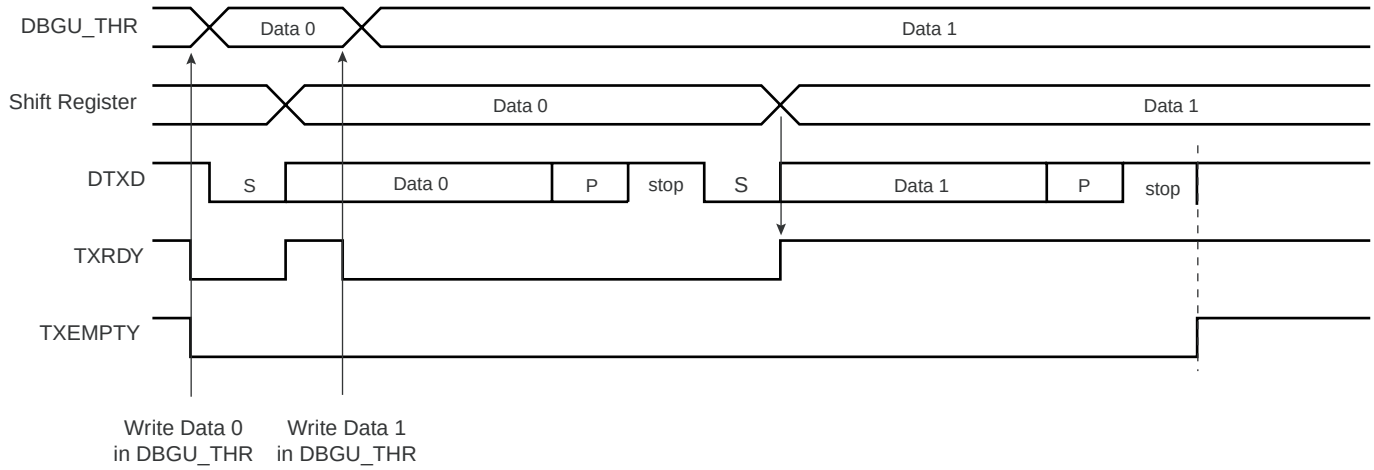
Transmitter Control

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 63. 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

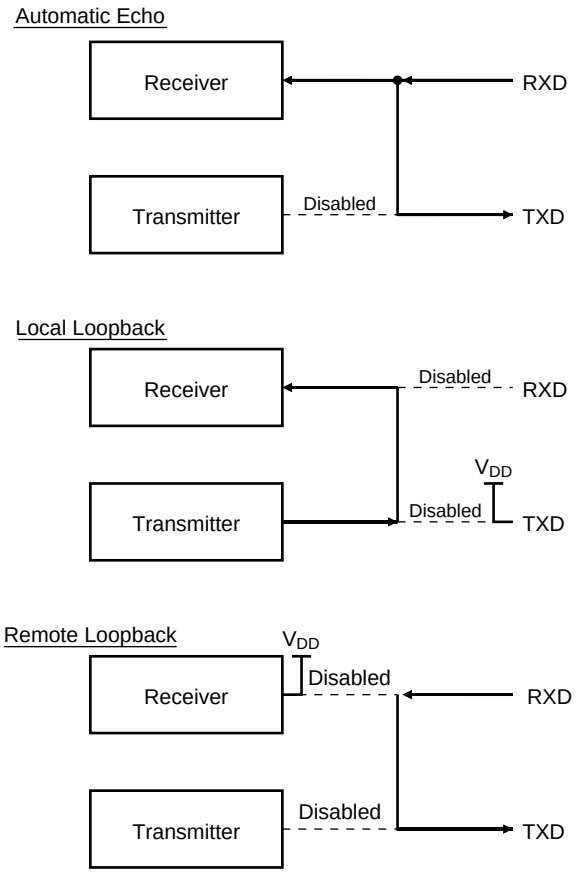
The Debug Unit supports three tests modes. These modes of operation are programmed by using the field CHMODE (Channel Mode) in the mode register DBGU_MR.

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 64. 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 35. 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 x 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
0	0	0	0	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

- **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	Reserved
1	0	1	1	Reserved
1	1	0	0	Reserved
1	1	0	1	Reserved
1	1	1	0	Reserved
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	Reserved
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		
Hex	**Dec**	**Architecture**
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

- **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

- **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 NTRST pin.

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 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.

Important features of the PIO also include:

- Up to 32 Programmable I/O Lines
- Fully Programmable through Set/Clear Registers
- Multiplexing of Two Peripheral Functions per I/O Line
- For each I/O Line (Whether Assigned to a Peripheral or Used as General Purpose I/O)
  - Input Change Interrupt
  - 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

Block Diagram

Figure 65. Block Diagram

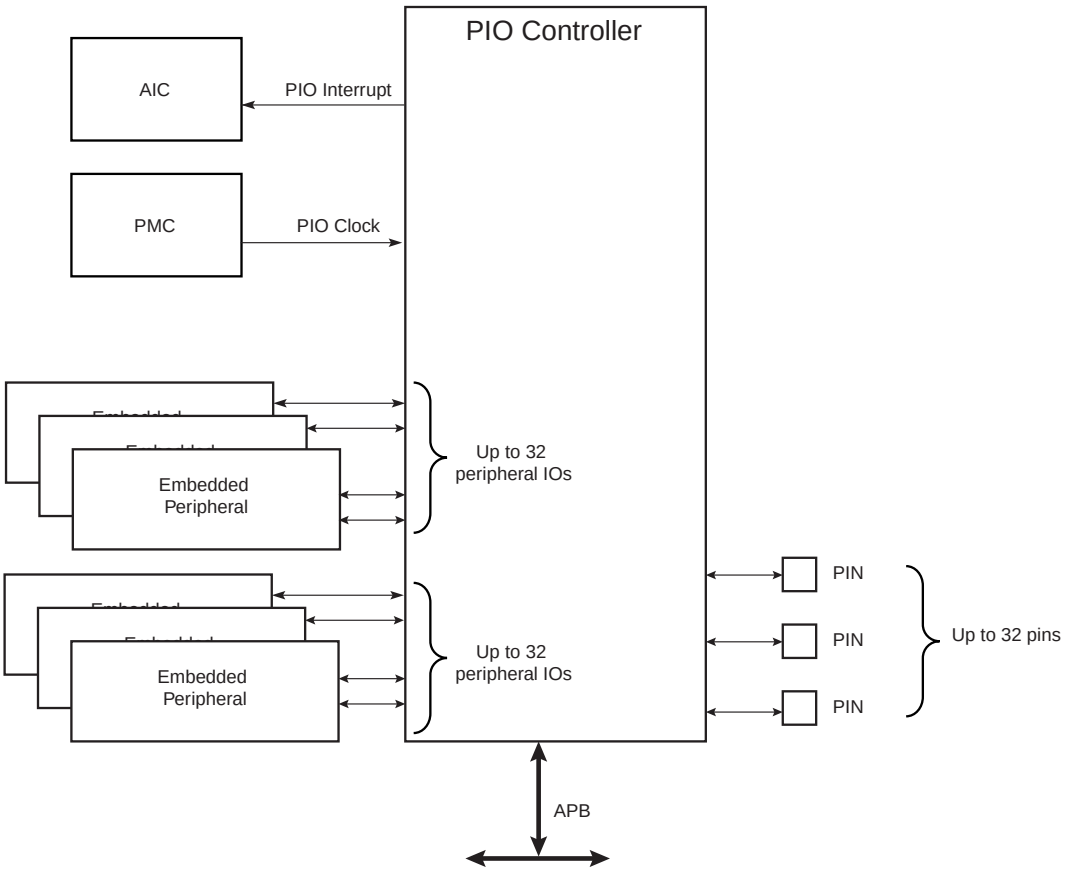
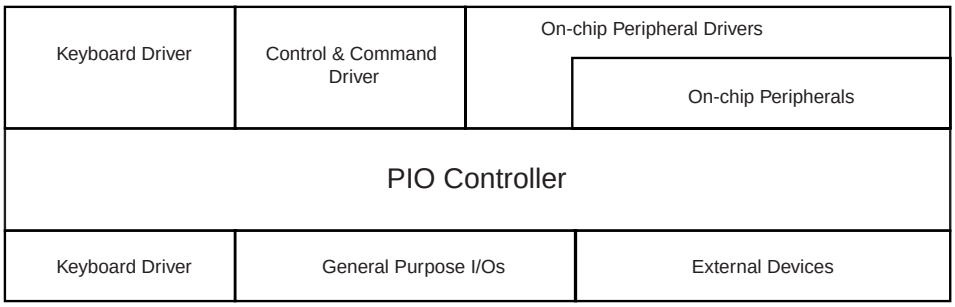


Figure 66. Application Block Diagram



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 (see Power Management Controller).

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

Interrupt Generation

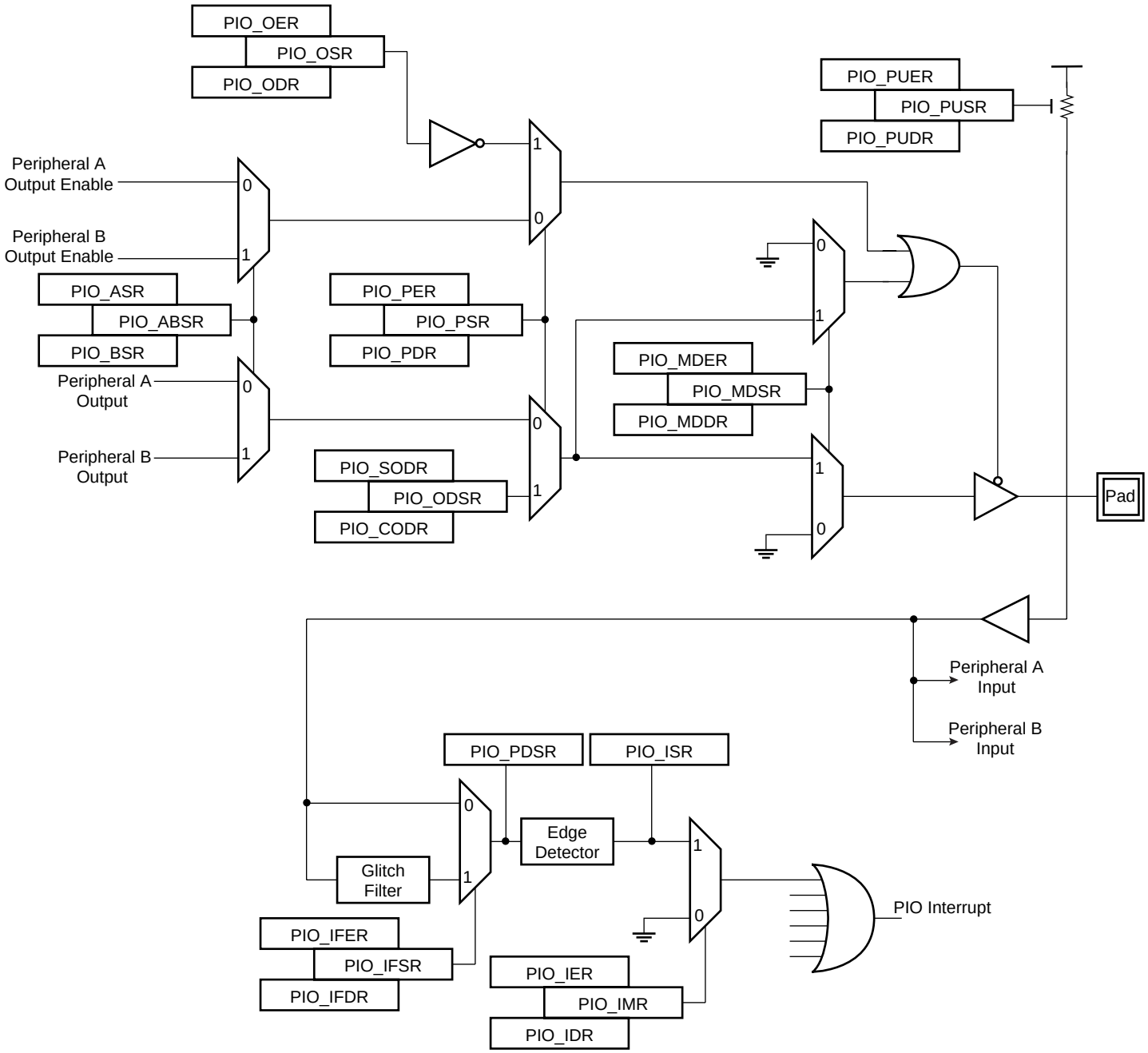
For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources 2 to 31. Refer to the PIO Controller peripheral identifier in the product description to identify the interrupt sources dedicated 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 67.

Figure 67. 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 kΩ (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 Resistor). Writing in these registers results in setting or clearing the corresponding bit in PIO_PUSR (Pull-up Status Register). Reading a 1 in PIO_PUSR means the pull-up is disabled and reading a 0 means the pull-up is enabled.

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/0 line is assigned to a peripheral function, i.e. the corresponding bit in PIO_PSR is at 0, the drive of the I/O line is controlled by the peripheral. Peripheral A or B, 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

Using the write operations in PIO_SODR and PIO_CODR can require that several instructions be executed in order to define values on several bits. Both clearing and setting I/O lines on an 8-bit port, for example, cannot be done at the same time, and thus might limit the application covered by the PIO Controller.

To avoid these inconveniences, the PIO Controller features a Synchronous Data Output to clear and set a number of I/O lines in a single write. This is performed by authorizing the writing of PIO_ODSR (Output Data Status Register) from the register set PIO_OWER (Output Write Enable Register), PIO_OWDR (Output Write Disable Register) and PIO_OWSR (Output Write Status Register). The value of PIO_OWSR register is user-definable by writing in PIO_OWER and PIO_OWDR. It is used by the PIO Controller as a PIO_ODSR write authorization mask. Authorizing the write of PIO_ODSR on a user-definable number of bits is especially useful, as it guarantees that the unauthorized bit will not be changed when writing it and thus avoids the need of a time consuming read-modify-write operation.

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.

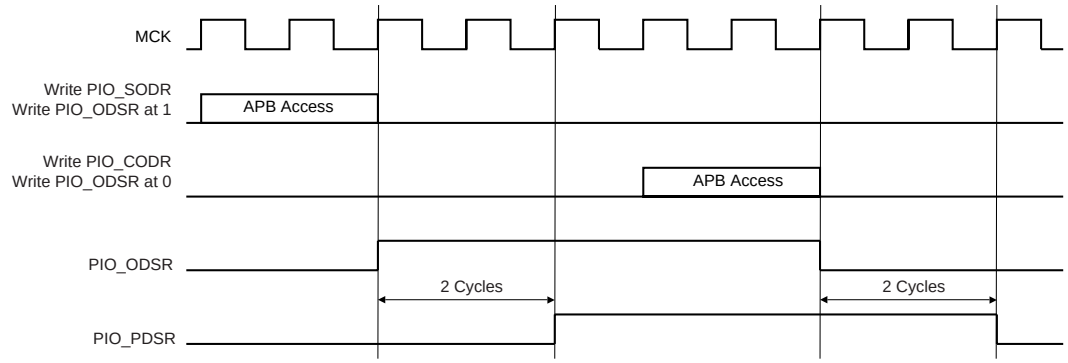
The Multi Drive feature is controlled by PIO_MDER (Multi-driver Enable Register) and PIO_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO_MDSR (Multi-driver Status Register) indicates 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 68 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 68 also shows when the feedback in PIO_PDSR is available.

Figure 68. 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.

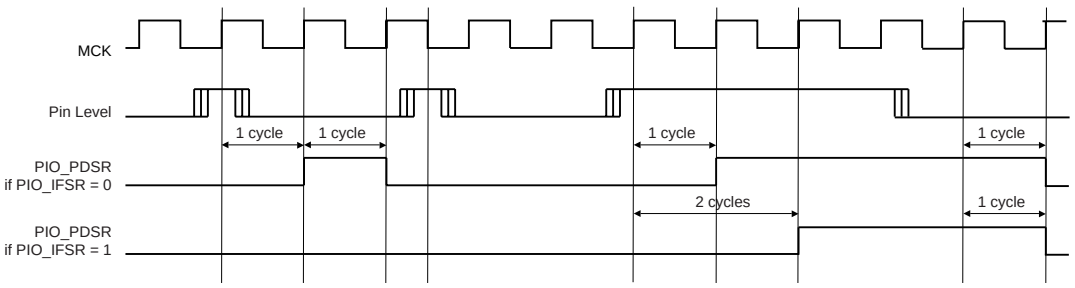
Input Glitch Filtering

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 69.

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 69. Input Glitch Filter Timing



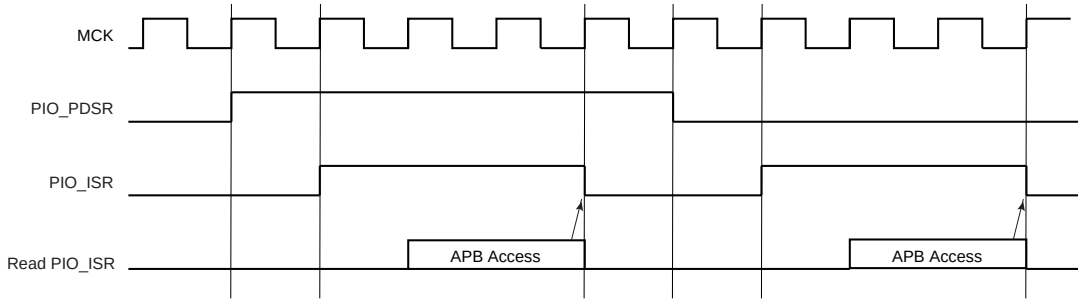
Input Change Interrupt

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 70. Input Change Interrupt Timings



I/O Lines Programming Example

The programing example shown in Table 36 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 lines 24 to 27 assigned to peripheral A with Input Change Interrupt and pull-up resistor

Table 36. 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 37. 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 (1)	PIO_PSR	Read-only	0x0000 0000
0x000C	Reserved			
0x0010	PIO Output Enable Register	PIO_OER	Write-only	–
0x0014	PIO Output Disable Register	PIO_ODR	Write-only	–
0x0018	PIO Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	PIO Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	PIO Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	PIO Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	PIO Set Output Data Register	PIO_SODR	Write-only	–
0x0034	PIO Clear Output Data Register	PIO_CODR	Write-only	–
0x0038	PIO Output Data Status Register(2)	PIO_ODSR	Read-only	0x0000 0000
0x003C	PIO Pin Data Status Register(3)	PIO_PDSR	Read-only	
0x0040	PIO Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	PIO Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	PIO Interrupt Mask Register	PIO_IMR	Read-only	0x0000 0000
0x004C	PIO Interrupt Status Register(4)	PIO_ISR	Read-only	0x0000 0000
0x0050	PIO Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	PIO Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	PIO Multi-driver Status Register	PIO_MDSR	Read-only	0x0000 0000
0x005C	Reserved			
0x0060	PIO Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	PIO Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	PIO Pad Pull-up Status Register	PIO_PUSR	Read-only	0x0000 0000
0x006C	Reserved			

Table 37. PIO Register Mapping (Continued)

Offset	Register	Name	Access	Reset Value
0x0070	PIO Peripheral A Select Register[5]	PIO_ASR	Write-only	–
0x0074	PIO Peripheral B Select Register[5]	PIO_BSR	Write-only	–
0x0078	PIO AB Status Register[5]	PIO_ABSR	Read-only	0x0000 0000
0x007C to 0x009C	Reserved			
0x00A0	PIO Output Write Enable	PIO_OWER	Write-only	–
0x00A4	PIO Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	PIO Output Write Status Register	PIO_OWSR	Read-only	0x0000 0000
0x00AC	Reserved			

Notes:
1. Reset value of PIO_PSR depends on the product implementation.
2. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
3. Reset value of PIO_PDSR depends on the level of the I/O lines.
4. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.
5. Only this set of registers clears the status by writing 1 in the first register and sets the status by writing 1 in the second register.

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 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 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 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 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 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 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 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 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 Filer Status**

0 = The input glitch filter is disabled on the I/O line.

1 = The input glitch filter is enabled on the I/O line.

PIO 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 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 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 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 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 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 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 Interrupt Status 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 = 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 Pad 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_ASR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24
23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16
15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8
7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Peripheral A Select**

0 = No effect.

1 = Assigns the I/O line to the Peripheral A function.

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 AB 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 allows communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the master that controls the data flow, while the other system acts as the slave, having data shifted into and out of it by the master. Different CPUs can take turn being masters (Multiple Master Protocol versus 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:

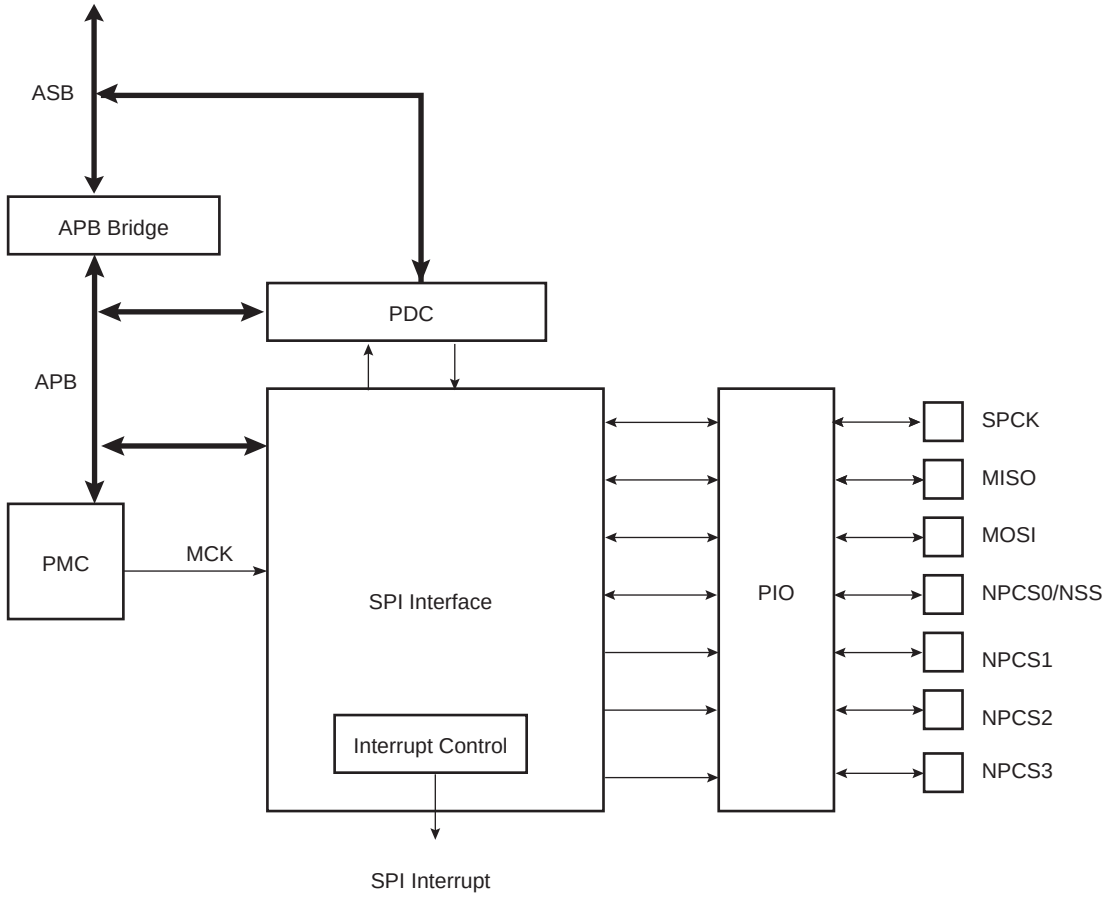
- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during 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.

The main features of the SPI are:

- Supports Communication with Serial External Devices
  - 4 Chip Selects with External Decoder Support Allow Communication with Up to 15 Peripherals
  - Serial Memories, such as DataFlash and 3-wire EEPROMs
  - Serial Peripherals, such as ADCs, DACs, LCD Controllers, CAN Controllers and Sensors
  - External Co-processors
- Master or Slave Serial Peripheral Bus Interface
  - 8- to 16-bit Programmable Data Length Per Chip Select
  - Programmable Phase and Polarity Per Chip Select
  - Programmable Transfer Delays Between Consecutive Transfers and Between Clock and Data Per Chip Select
  - Programmable Delay Between Consecutive Transfers
  - Selectable Mode Fault Detection
- Connection to PDC Channel Capabilities Optimizes Data Transfers
  - One Channel for the Receiver, One Channel for the Transmitter
  - Next Buffer Support

Block Diagram

Figure 71. Block Diagram



Application Block Diagram

Figure 72. Application Block Diagram: Single Master/Multiple Slave Implementation

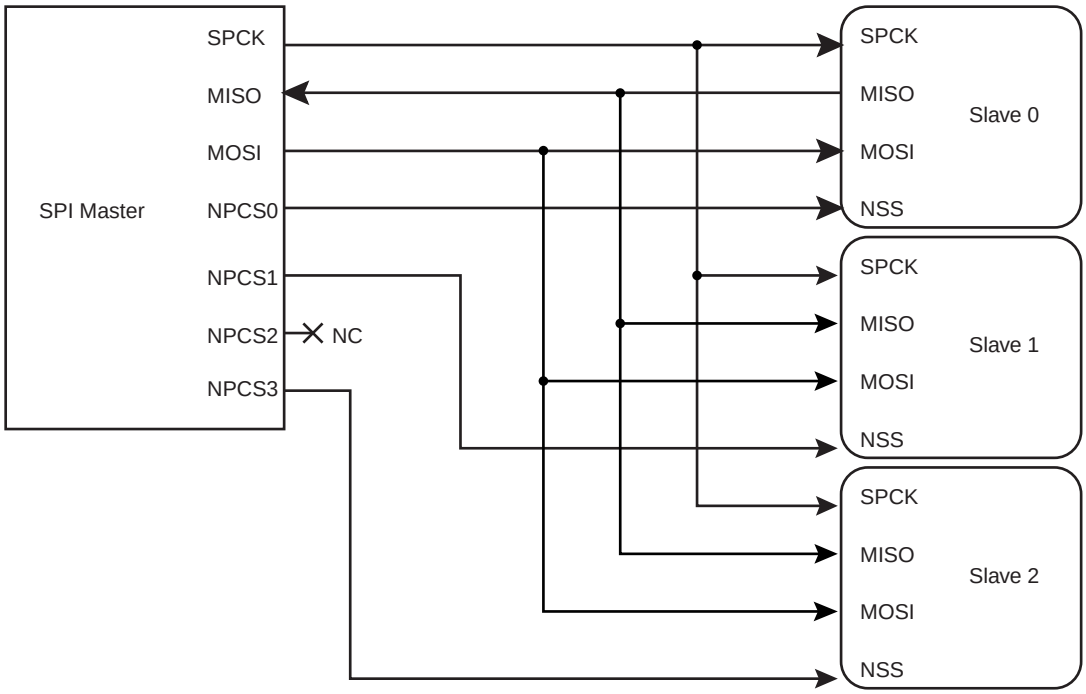


Table 38. Signal Description

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 have to 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

Master Mode Operations

When configured in Master Mode, the Serial Peripheral Interface controls data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select(s) to the slave(s) and the serial clock (SPCK). After enabling the SPI, a data transfer begins when the core writes to the SPI_TDR (Transmit Data Register).

Transmit and Receive buffers maintain the data flow at a constant rate with a reduced requirement for high-priority interrupt servicing. When new data is available in the SPI_TDR, the SPI continues to transfer data. If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error (OVRES) flag is set.

Note: As long as this flag is set, no data is loaded in the SPI_RDR. The user has to read the status register to clear it.

The programmable delay between the activation of the chip select and the start of the data transfer (DLYBS), as well as the delay between each data transfer (DLYBCT), can be programmed for each of the four external chip selects. All data transfer characteristics, including the two timing values, are programmed in registers SPI_CSR0 to SPI_CSR3 (Chip Select Registers).

In Master Mode, the peripheral selection can be defined 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

Figure 77 and Figure 78 show the operation of the SPI in Master Mode. For details concerning the flag and control bits in these diagrams, see the tables in the Programmer's Model, starting in Section .

Fixed Peripheral Select

This mode is used for transferring memory blocks without the extra overhead in the transmit data register to determine the peripheral.

Fixed Peripheral Select is activated by setting bit PS to zero in SPI_MR (Mode Register). The peripheral is defined by the PCS field in SPI_MR.

This option is only available when the SPI is programmed in Master Mode.

Variable Peripheral Select

Variable Peripheral Select is activated by setting bit PS to one. The PCS field in SPI_TDR is used to select the destination peripheral. The data transfer characteristics are changed when the selected peripheral changes, according to the associated chip select register.

The PCS field in the SPI_MR has no effect.

This option is only available when the SPI is programmed in Master Mode.

Chip Selects

The Chip Select lines are driven by the SPI only if it is programmed in Master Mode. These lines are used to select the destination peripheral. The PCSDEC field in SPI_MR (Mode Register) selects one to four peripherals (PCSDEC = 0) or up to 15 peripherals (PCSDEC = 1).

If Variable Peripheral Select is active, the chip select signals are defined for each transfer in the PCS field in SPI_TDR. Chip select signals can thus be defined independently for each transfer.

If Fixed Peripheral Select is active, Chip Select signals are defined for all transfers by the field PCS in SPI_MR. If a transfer with a new peripheral is necessary, the software must wait until the current transfer is completed, then change the value of PCS in SPI_MR before writing new data in SPI_TDR.

The value on the NPCS pins at the end of each transfer can be read in the SPI_RDR (Receive Data Register).

By default, all NPCS signals are high (equal to one) before and after each transfer.

Clock Generation and Transfer Delays

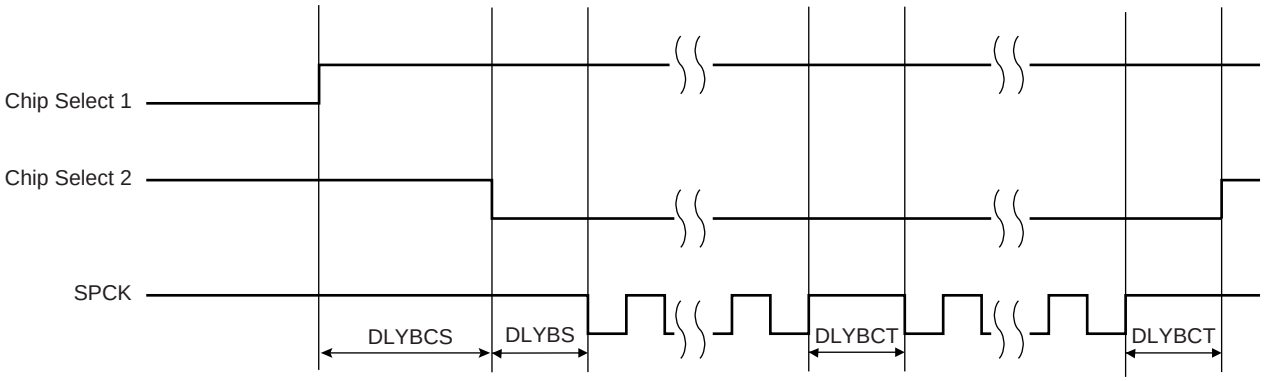
The SPI Baud rate clock is generated by dividing the Master Clock (MCK) or the Master Clock divided by 32 (if DIV32 is set in the Mode Register) by a value between 4 and 510. The divisor is defined in the SCBR field in each Chip Select Register. The transfer speed can thus be defined independently for each chip select signal.

Figure 73 shows a chip select transfer change and consecutive transfers on the same chip selects. Three delays can be programmed to modify the transfer waveforms:

- Delay between chip selects, programmable only once for all the chip selects by writing the field DLYBCS in the Mode Register. Allows insertion of a delay between release of one chip select and before assertion of a new one.
- Delay before SPCK, independently programmable for each chip select by writing the field DLYBS. Allows the start of SPCK to be delayed until after the chip select has been asserted.
- Delay between consecutive transfers, independently programmable for each chip select by writing the field DLYBCT. 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 73. Programmable Delays



Mode Fault Detection

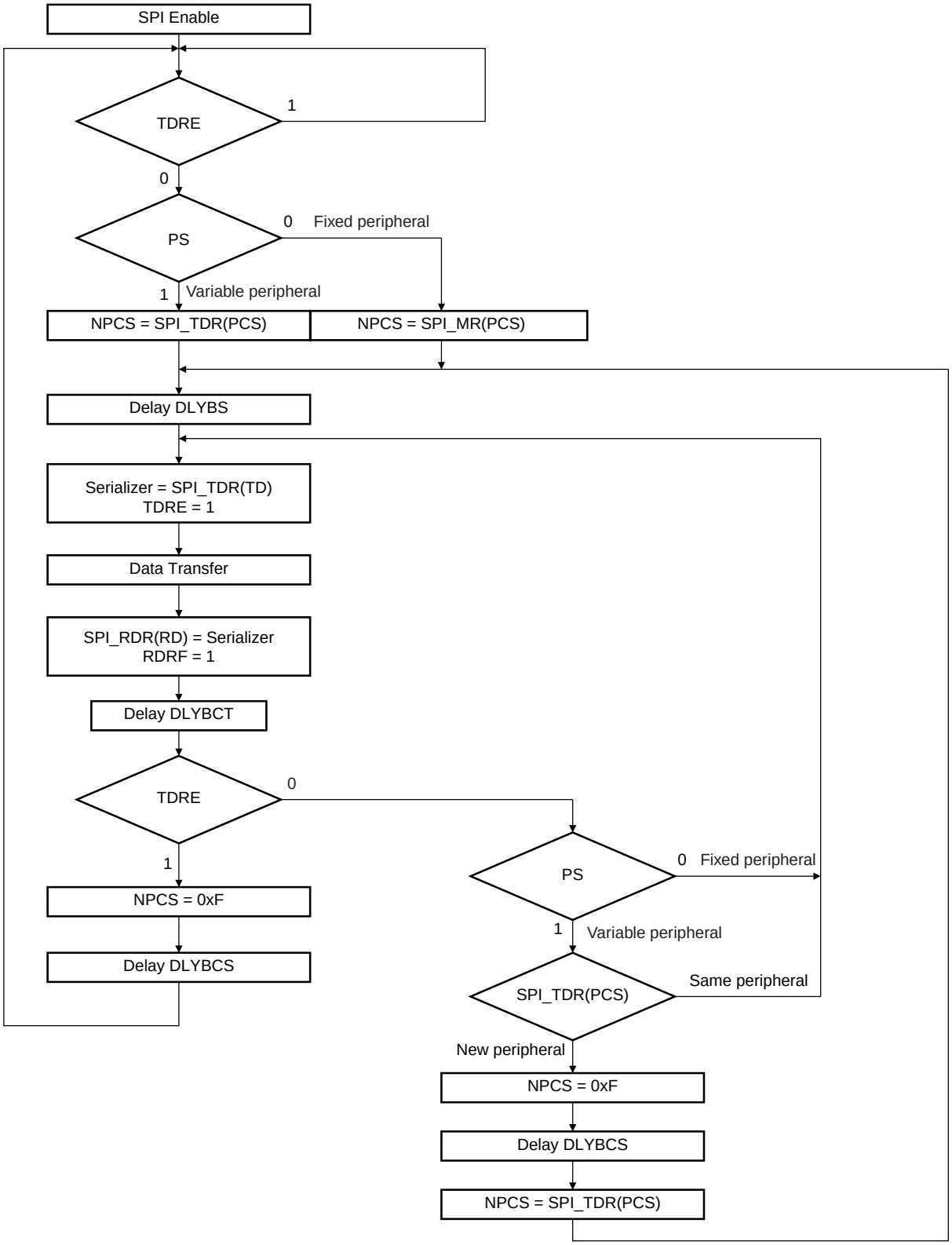
A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the NPCS0/NSS signal.

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 disabled until re-enabled by bit SPIEN in the SPI_CR (Control Register).

By default, Mode Fault Detection is enabled. It is disabled by setting the MODFDIS bit in the SPI Mode Register.

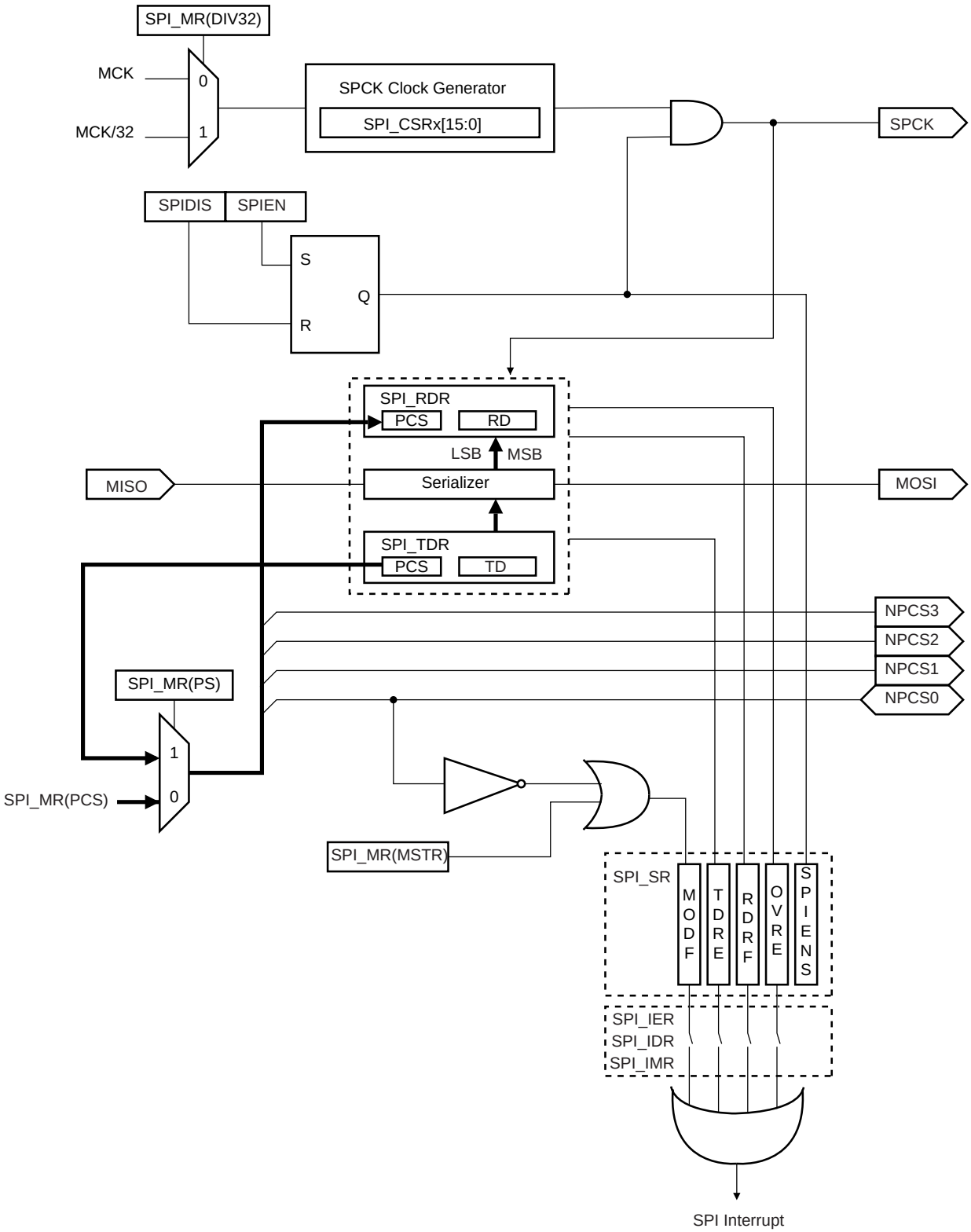
Master Mode Flow Diagram

Figure 74. Master Mode Flow Diagram



Master Mode Block Diagram

Figure 75. Master Mode Block Diagram



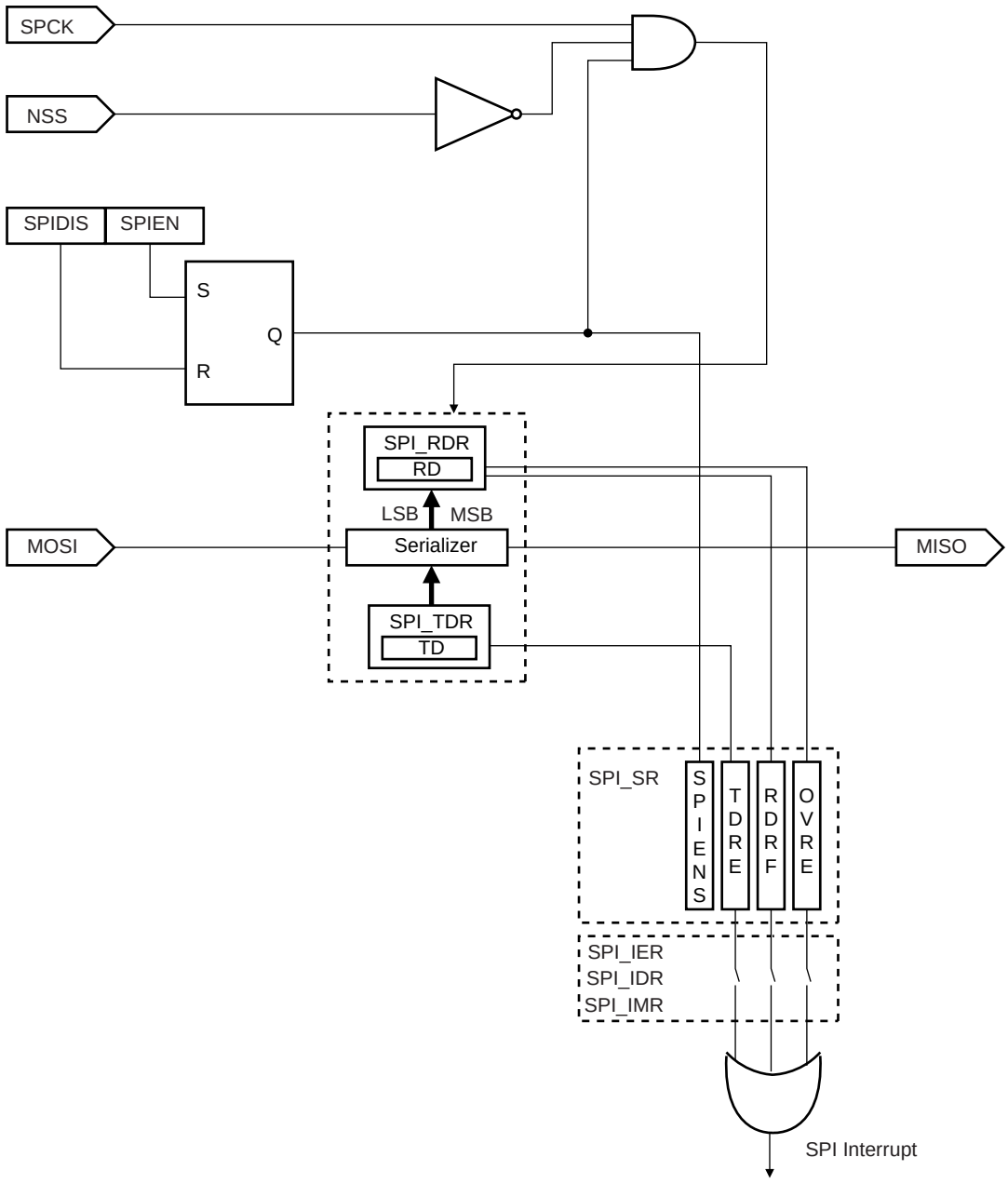
SPI Slave Mode

In Slave Mode, the SPI waits for NSS to go active low before receiving the serial clock from an external master.

In Slave Mode, CPOL, NCPHA and BITS fields of SPI_CSR0 are used to define the transfer characteristics. The other Chip Select Registers are not used in Slave Mode.

In Slave Mode, the low and high pulse durations of the input clock on SPCK must be longer than two Master Clock periods.

Figure 76. Slave Mode Block Diagram



Data Transfer

Four modes are used for data transfers. These modes correspond to combinations of a pair of parameters called clock polarity (CPOL) and clock phase (NCPHA) that determine the edges of the clock signal on which the data are 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.

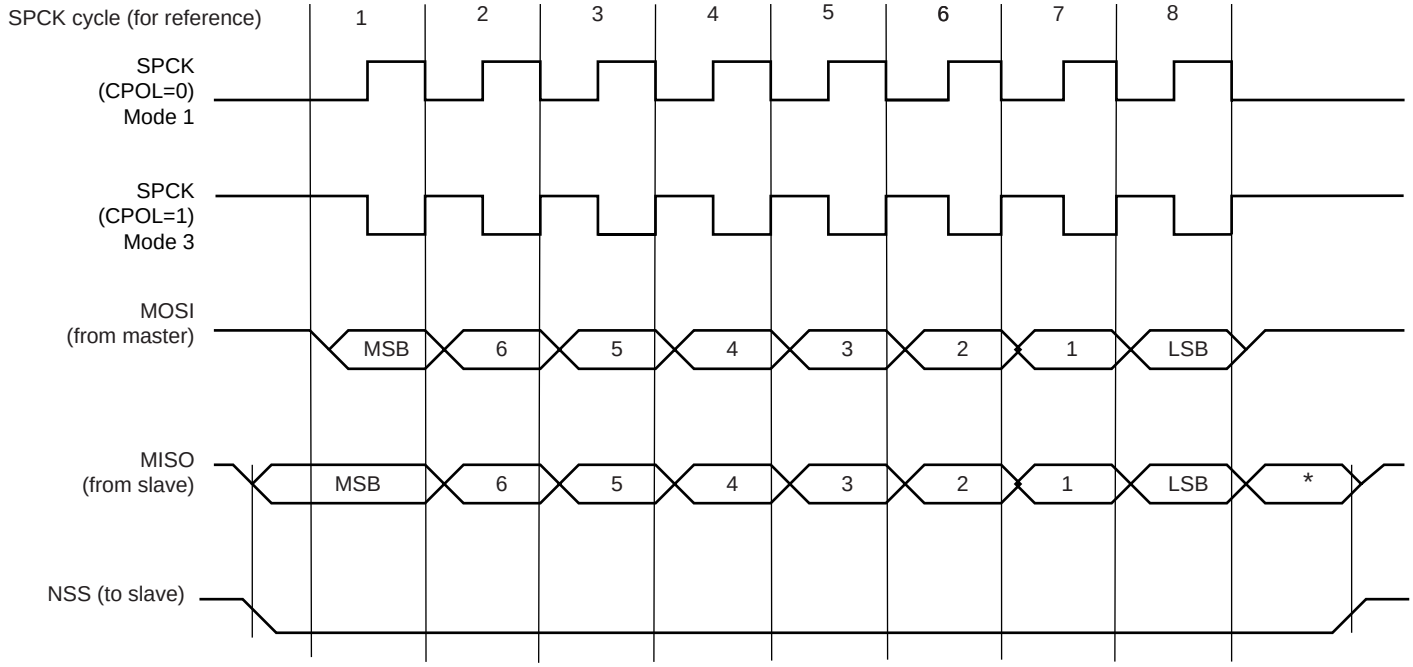
Table 39 shows the four modes and corresponding parameter settings.

Table 39. SPI Bus Protocol Mode

SPI Mode	CPOL	NCPHA
0	0	0
1	0	1
2	1	0
3	1	1

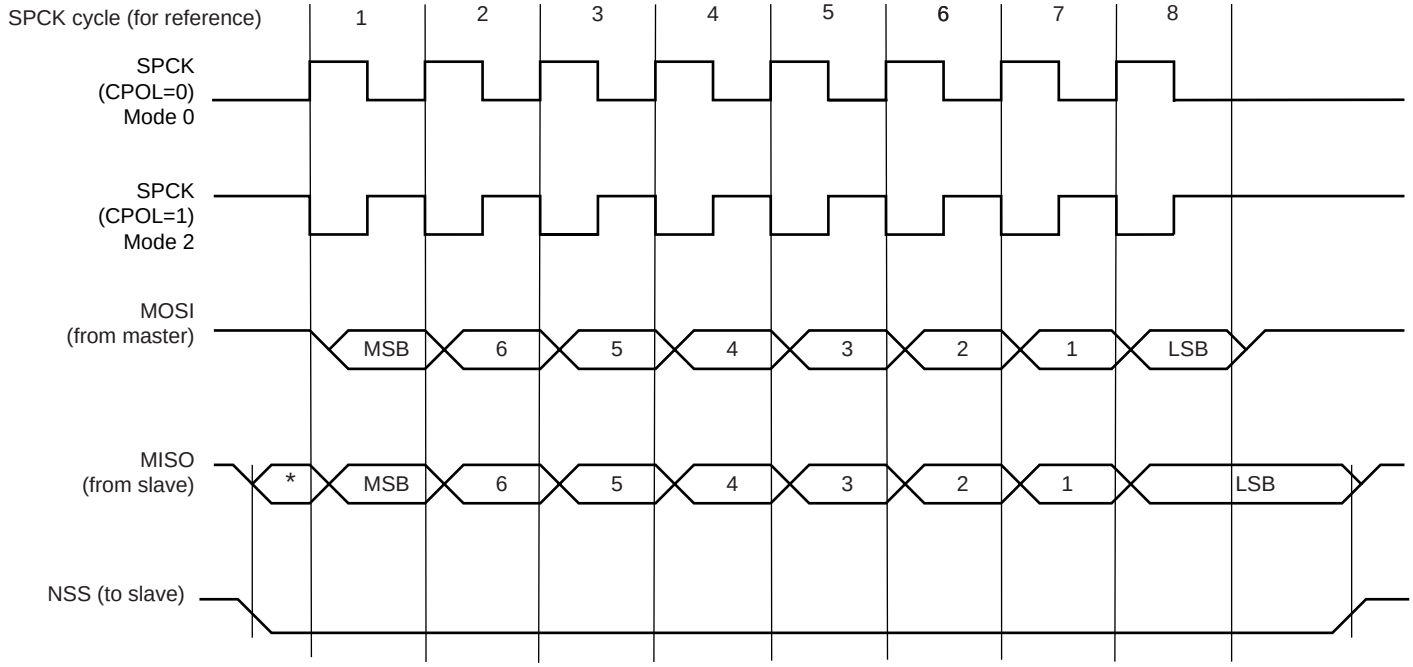
Figure 77 and Figure 78 show examples of data transfers.

Figure 77. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



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

Figure 78. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined but normally LSB of previous character transmitted.

Serial Peripheral Interface (SPI) User Interface

Table 40. 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
0x40 - 0xFF	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
–	–	–	–	–	–	–	–
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.

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	DIV32	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 16 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*.

***Note**: The 16th state corresponds to a state in which all chip selects are inactive. This allows a different clock configuration to be defined by each chip select register*.

- **DIV32: Clock Selection**

0 = The SPI operates at MCK.

1 = The SPI operates at MCK/32.

- **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 192 MCK periods if DIV32 is set) will be inserted by default.

Otherwise, the following equation determines the delay:

If DIV32 is 0:

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

If DIV32 is 1:

$$\text{Delay Between Chip Selects} = \text{DLYBCS} \times 32 / \text{MCK}$$

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
–	–	–	–	–	–	–	–
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

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
–	–	–	–	–	–	–	–
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 have a value other than 0.

1 = Both SPI_RCR and SPI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = SPI_TCR or SPI_TNCR have a value other than 0.

1 = Both SPI_TCR and SPI_TNCR have a value of 0.

- **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
–	–	–	–	–	–	–	–
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**

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
–	–	–	–	–	–	–	–
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**

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
–	–	–	–	–	–	–	–
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**

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				–	–	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.

- **BITS: Bits Per Transfer**

The BITS field determines the number of data bits transferred. Reserved values should not be used.

BITS[3:0]	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 2 to 255 in the field SCBR. The following equation determines the SPCK baud rate:

If DIV32 is 0:

$$\text{SPCK Baudrate} = \text{MCK} / (2 \times \text{SCBR})$$

If DIV32 is 1:

$$\text{SPCK Baudrate} = \text{MCK} / (64 \times \text{SCBR})$$

Giving SCBR a value of zero or one disables the baud rate generator. SPCK is disabled and assumes its inactive state value. No serial transfers may occur. At reset, baud rate is disabled.

- **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 DIV32 is 0:

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

If DIV32 is 1:

$$\text{Delay Before SPCK} = 32 \times \text{DLYBS} / \text{MCK}$$

- **DLYBCT: Delay Between Consecutive Transfers**

This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select. The delay is always inserted after each transfer and before removing the chip select if needed.

When DLYBCT equals zero, a minimum delay of four MCK cycles are inserted (or 128 MCK cycles when DIV32 is set) between two consecutive characters.

Otherwise, the following equation determines the delay:

If DIV32 is 0:

$$\text{Delay Between Consecutive Transfers} = 32 \times \text{DLYBCT} / \text{MCK}$$

If DIV32 is 1:

$$\text{Delay Between Consecutive Transfers} = 1024 \times \text{DLYBCT} / \text{MCK}$$

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.

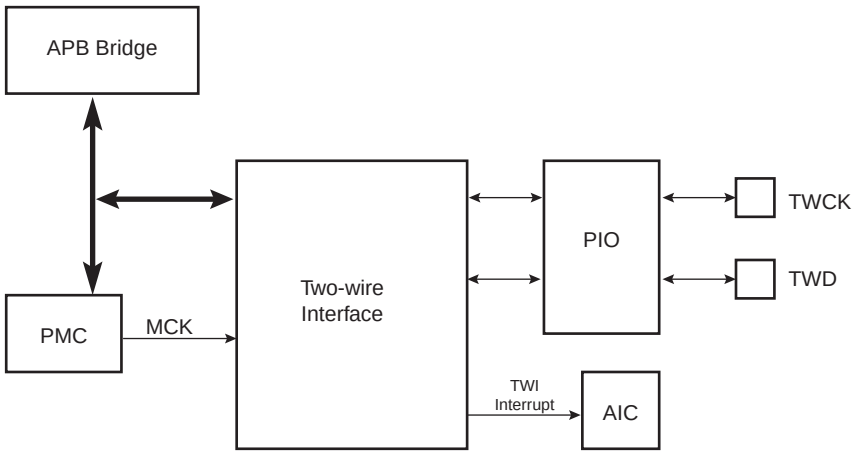
A configurable baud rate generator permits the output data rate to be adapted to a wide range of core clock frequencies.

The main features of the TWI are:

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

Block Diagram

Figure 79. Block Diagram



Application Block Diagram

Figure 80. Application Block Diagram

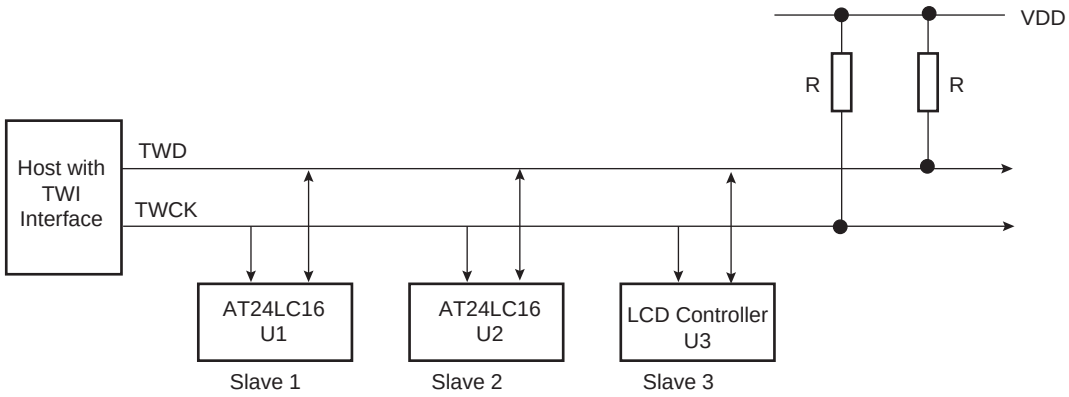


Table 41. I/O Lines Description

Pin Name	Pin Description	Type
TWD	Two-wire Serial Data	Input/Output
TWCK	Two-wire Serial Clock	Input/Output

Product Dependencies

I/O Lines

Both TWD and TWCK are bi-directional lines, connected to a positive supply voltage via a current source or pull-up resistor (see Figure 80 on page 251). 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.

Power Management

- Enable the peripheral clock.

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

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 eight 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 82 on page 253).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 81 on page 252).

- 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 81. START and STOP Conditions

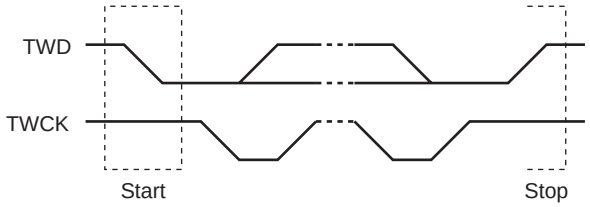
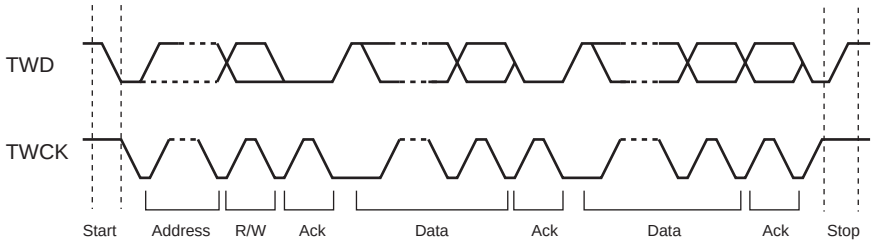


Figure 82. Transfer Format



Modes of Operation

The TWI has two modes of operations:

- 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 84 on page 254). 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, the **IADRSZ** must be set to 0. For slave address higher than seven bits, the user must configure the address size **(IADRSZ**) and set the other slave address bits in the internal address register (TWI_IADR).

Figure 83. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

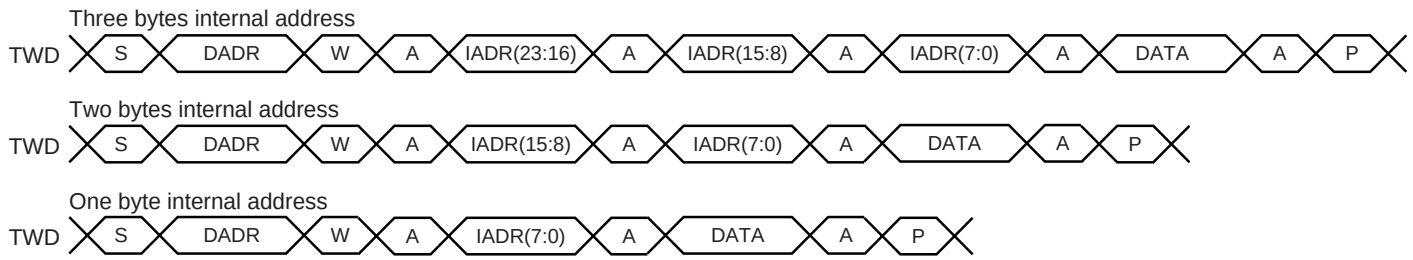


Figure 84. Master Write with One Byte Internal Address and Multiple Data Bytes

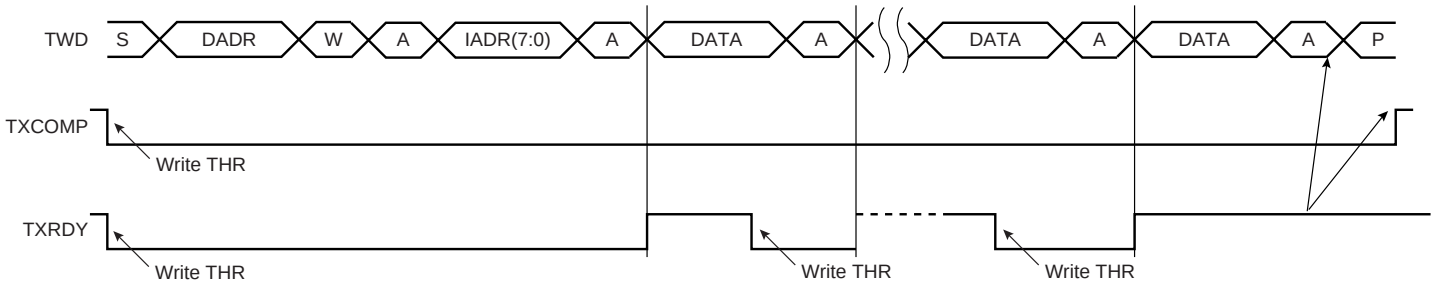


Figure 85. Master Read with One, Two or Three Bytes Internal Address and One Data Byte

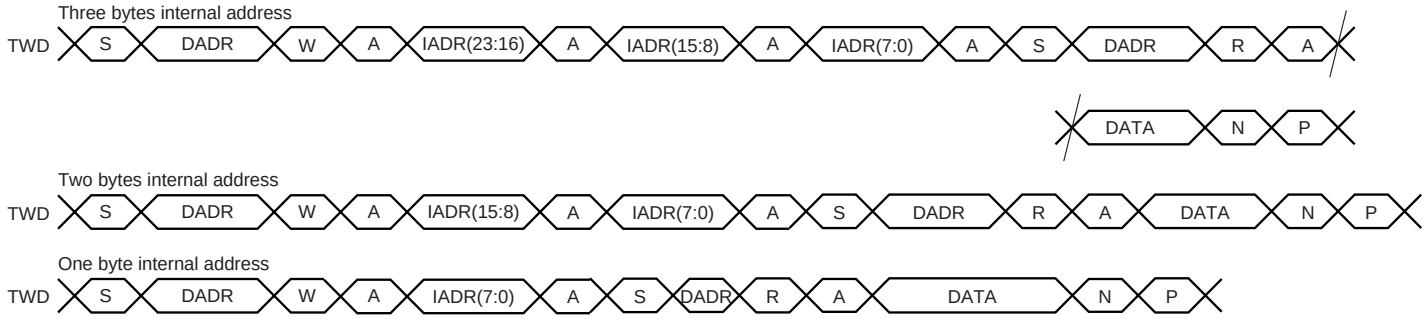
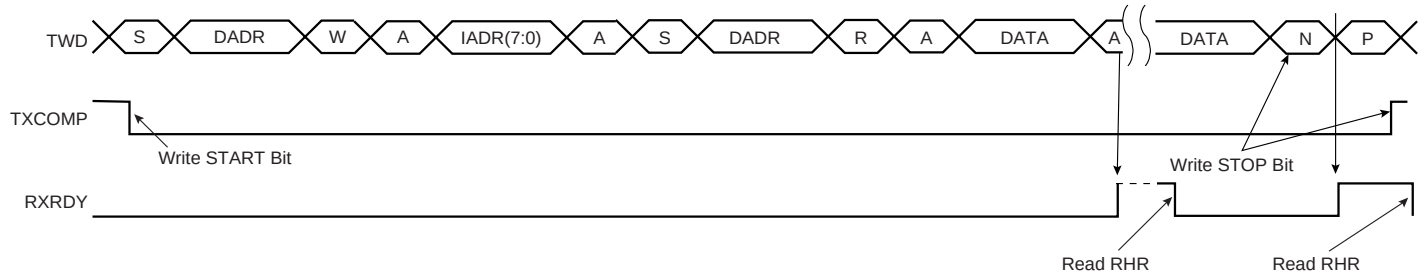


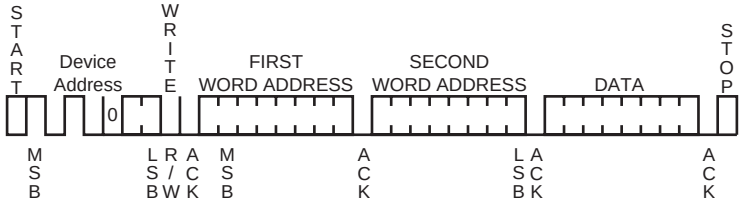
Figure 86. Master Read with One Byte Internal Address and Multiple Data Bytes



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

Figure 87 shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 87. Internal Address Usage



Read/Write Flowcharts

The following flowcharts shown in Figure 88 on page 256 and in Figure 89 on page 257 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 88. TWI Write in Master Mode

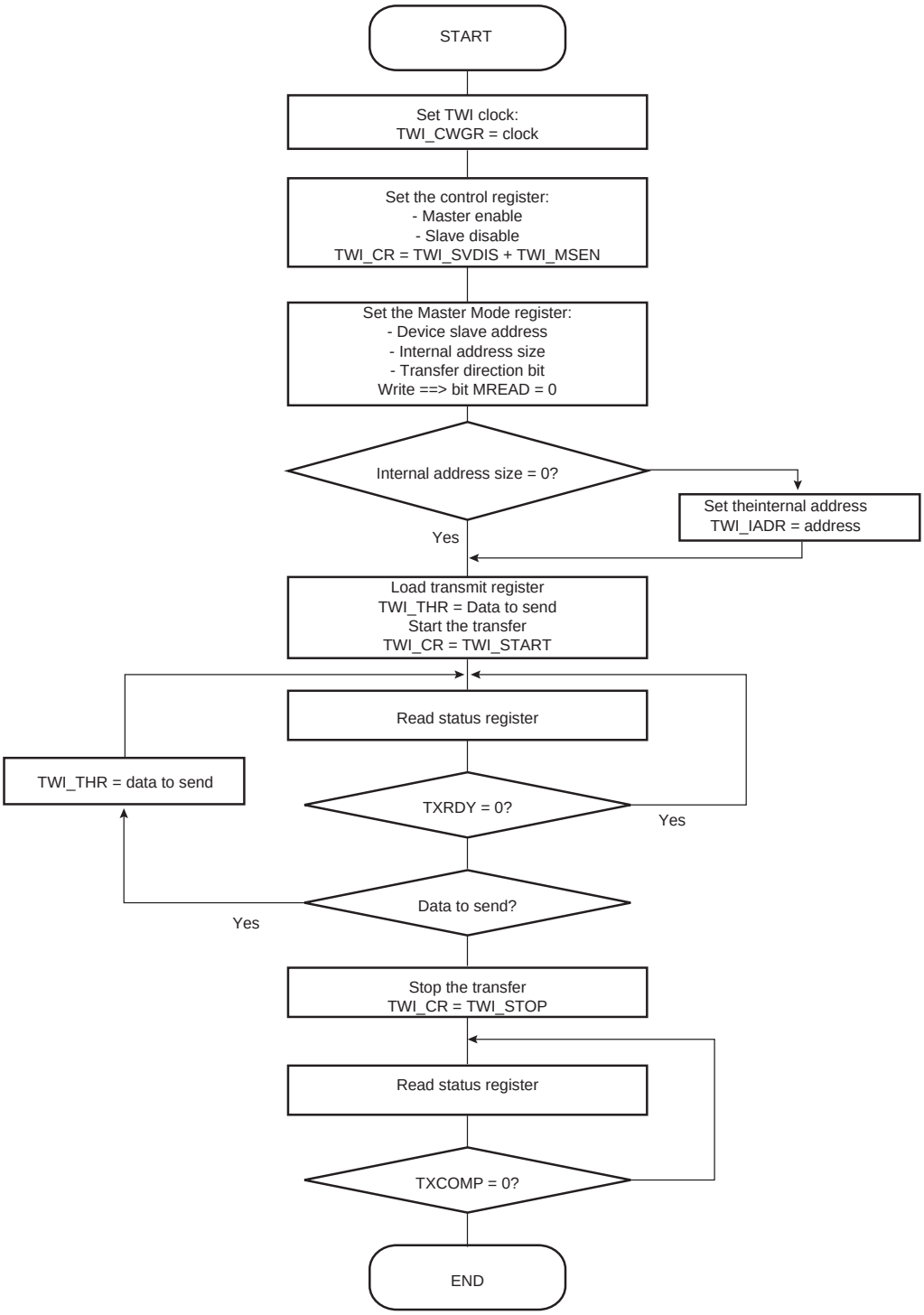
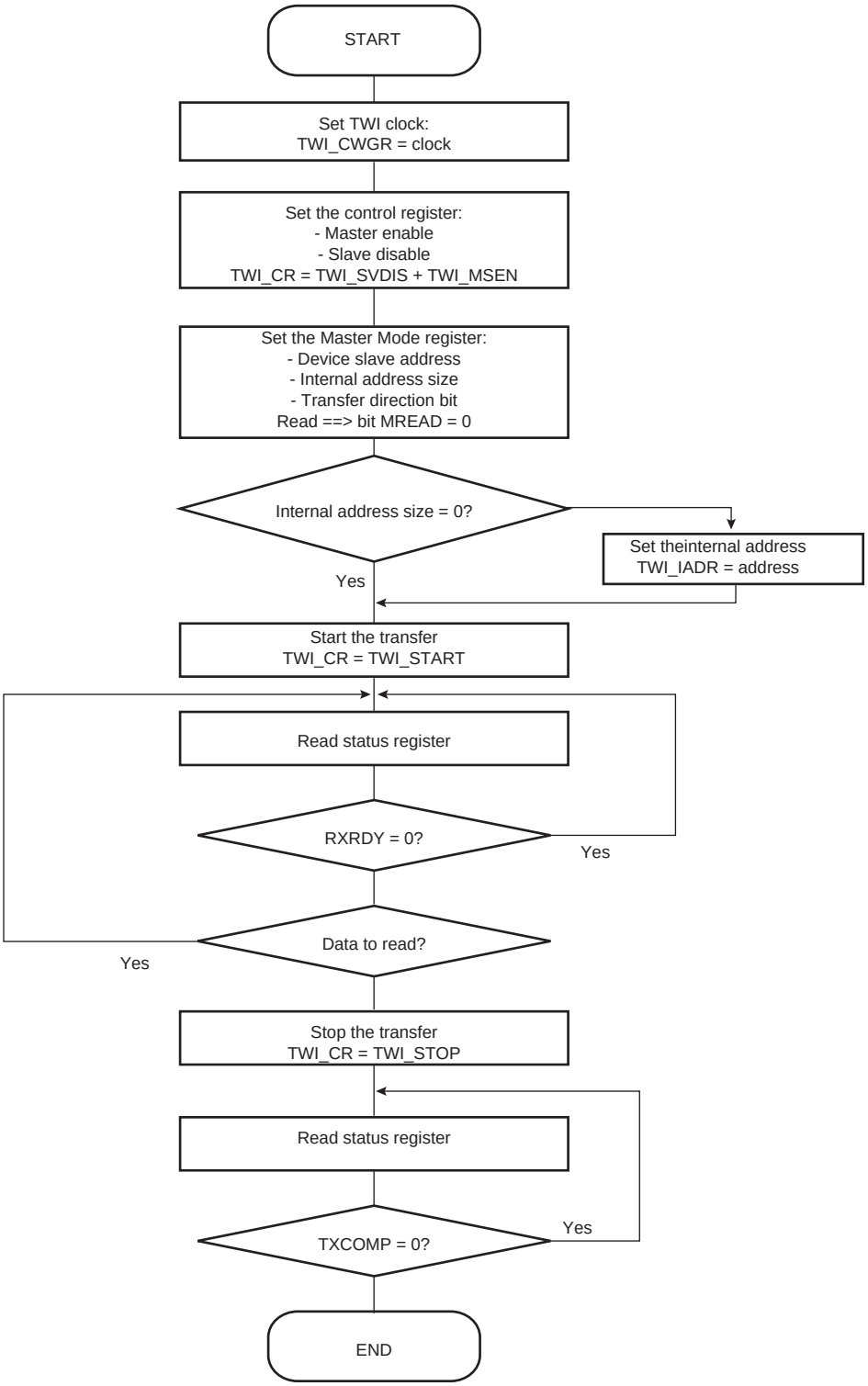


Figure 89. TWI Read in Master Mode



Two-wire Interface (TWI) User Interface

Table 42. 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

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 it contains 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 TWCK low period is defined as follows:

$$T_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 3) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The TWCK high period is defined as follows:

$$T_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 3) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both TWCK 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 shifter 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 theTWI_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 Transceiver (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 time-out 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 busses, 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.

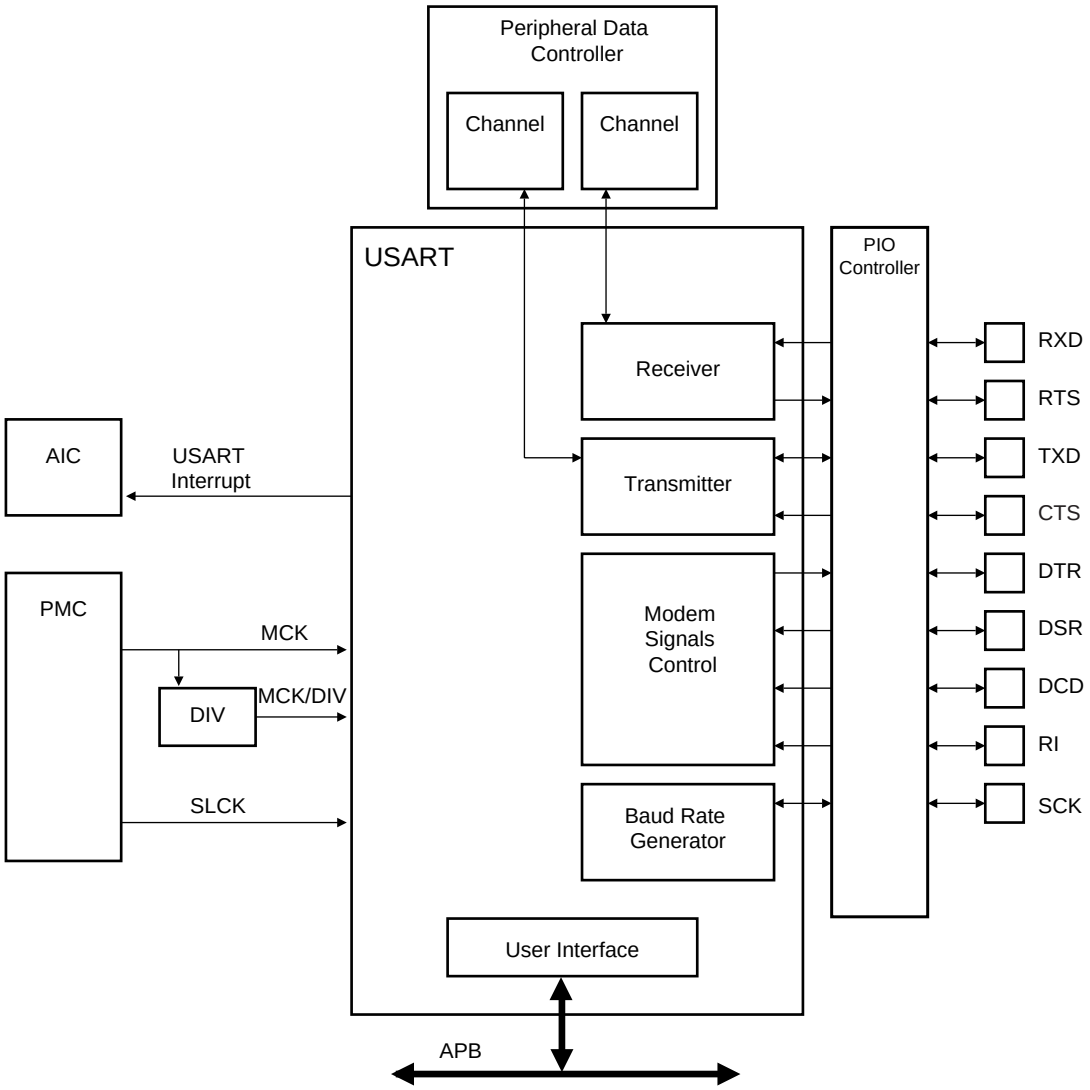
The USART supports the connection to the Peripheral Data Controller, which enables data transfers to the transmitter and from the receiver. The PDC provides chained buffer management without any intervention of the processor.

Important features of the USART are:

- Programmable Baud Rate Generator
- 5- to 9-bit Full-duplex Synchronous or Asynchronous Serial Communications
  - 1, 1.5 or 2 Stop Bits in Asynchronous Mode or 1 or 2 Stop Bits in Synchronous Mode
  - Parity Generation and Error Detection
  - Framing Error Detection, Overrun Error Detection
  - MSB- or LSB-first
  - Optional Break Generation and Detection
  - By 8 or by-16 Over-sampling Receiver Frequency
  - Optional Hardware Handshaking RTS-CTS
  - Optional Modem Signal Management DTR-DSR-DCD-RI
  - Receiver Time-out and Transmitter Timeguard
  - Optional Multi-Drop Mode with Address Generation and Detection
- RS485 with driver control signal
- ISO7816, T = 0 or T = 1 Protocols for Interfacing with Smart Cards
  - NACK Handling, Error Counter with Repetition and Iteration Limit
- IrDA Modulation and Demodulation
  - Communication at up to 115.2 Kbps
- Test Modes
  - Remote Loopback, Local Loopback, Automatic Echo
- Supports Connection of Two Peripheral Data Controller Channels (PDC)
  - Offer Buffer Transfer without Processor Intervention

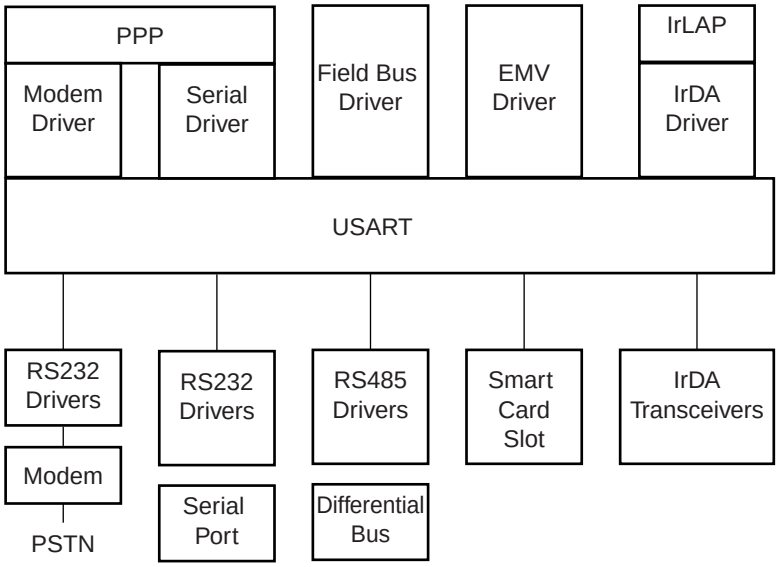
Block Diagram

Figure 90. USART Block Diagram



Application Block Diagram

Figure 91. Application Block Diagram



I/O Lines Description

Table 43. 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 not be implemented on the USART within a product. Frequently, only the USART1 is fully equipped with all the modem signals. For the other USARTs of the product not equipped with the corresponding pin, the associated control bits and statuses have no effect on the behavior of the USART.

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.

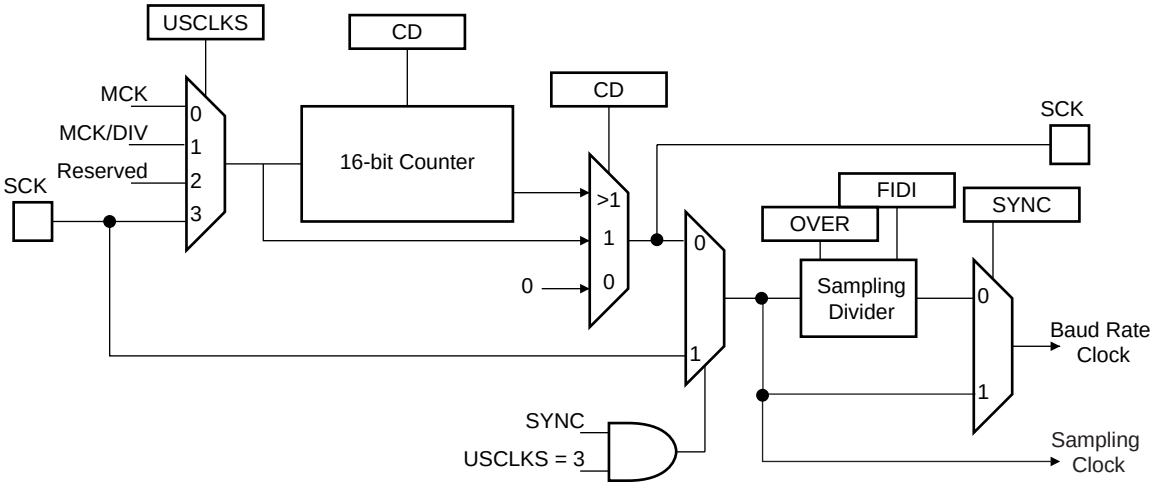
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US_MR) between:

- 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 92. 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 44 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 44. 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 44. 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 45.

Table 45. 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 46.

Table 46. 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 47 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock..

Table 47. 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

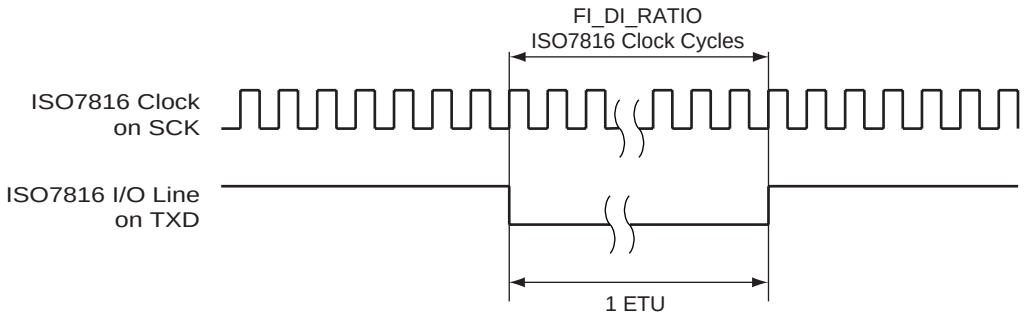
If the USART is configured in ISO7816 Mode, the clock selected by the USCLKS field in the Mode Register (US_MR) is first divided by the value programmed in the field CD in the Baud Rate Generator Register (US_BRGR). The resulting clock can be provided to the SCK pin to feed the smart card clock inputs. 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 93 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 93. 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 time guard is programmed, it is handled normally.

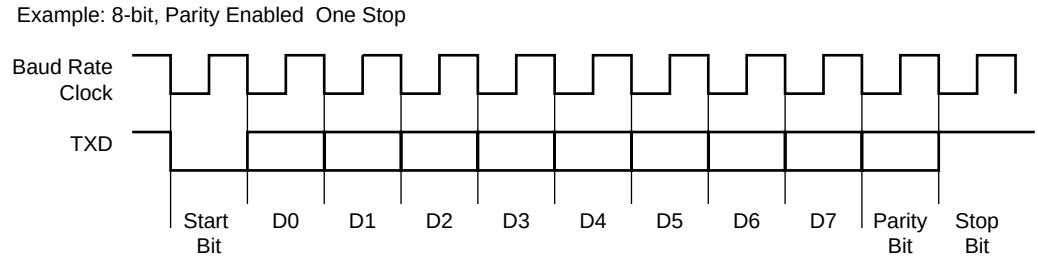
Synchronous and Asynchronous Modes

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.

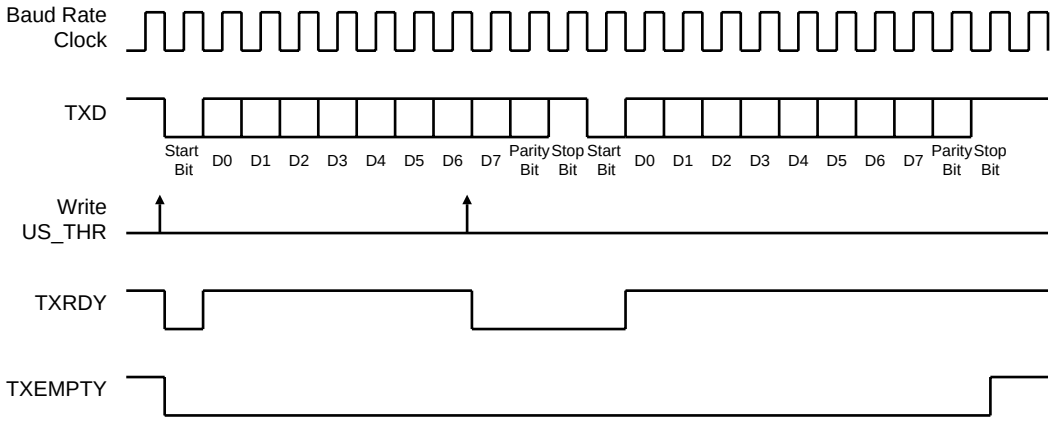
Figure 94. Character Transmit



The characters are sent by writing in the Transmit Holding Register (US_THR). The transmitter reports two status bits in the Channel Status Register (US_CSR): TXRDY (Transmitter Ready), which indicates that US_THR is empty and TXEMPTY, which indicates that all the characters written in US_THR have been processed. When the current character processing is completed, the last character written in US_THR is transferred into the Shift Register of the transmitter and US_THR becomes empty, thus TXRDY 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 95. 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 96 and Figure 97 illustrate start detection and character reception when USART operates in asynchronous mode.

Figure 96. Asynchronous Start Detection

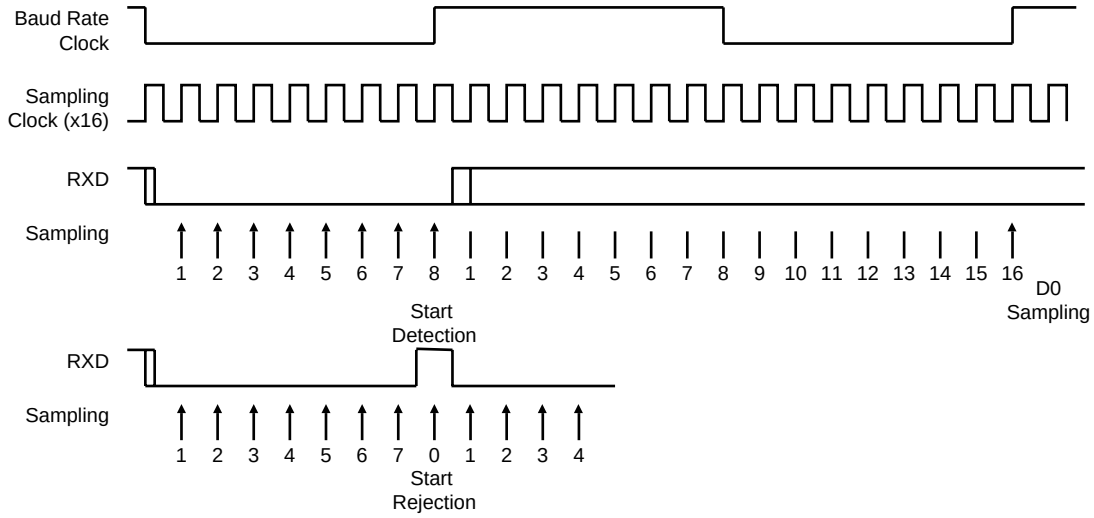
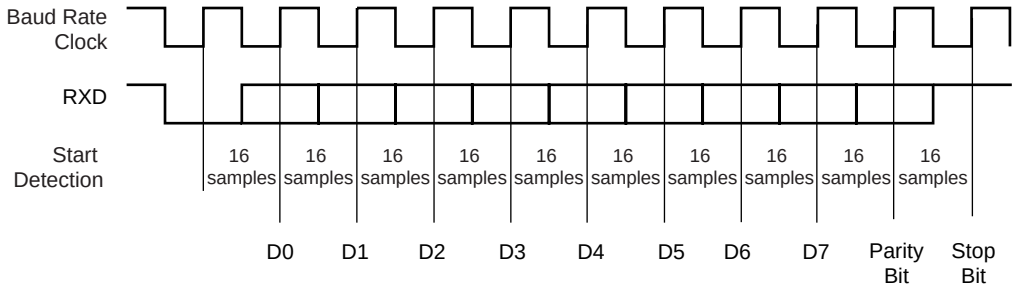


Figure 97. Asynchronous Character Reception

Example: 8-bit, Parity Enabled



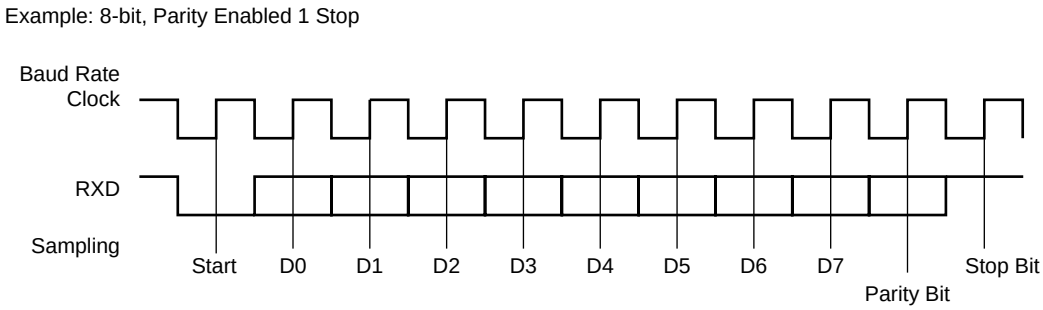
Synchronous Receiver

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 98 illustrates a character reception in synchronous mode.

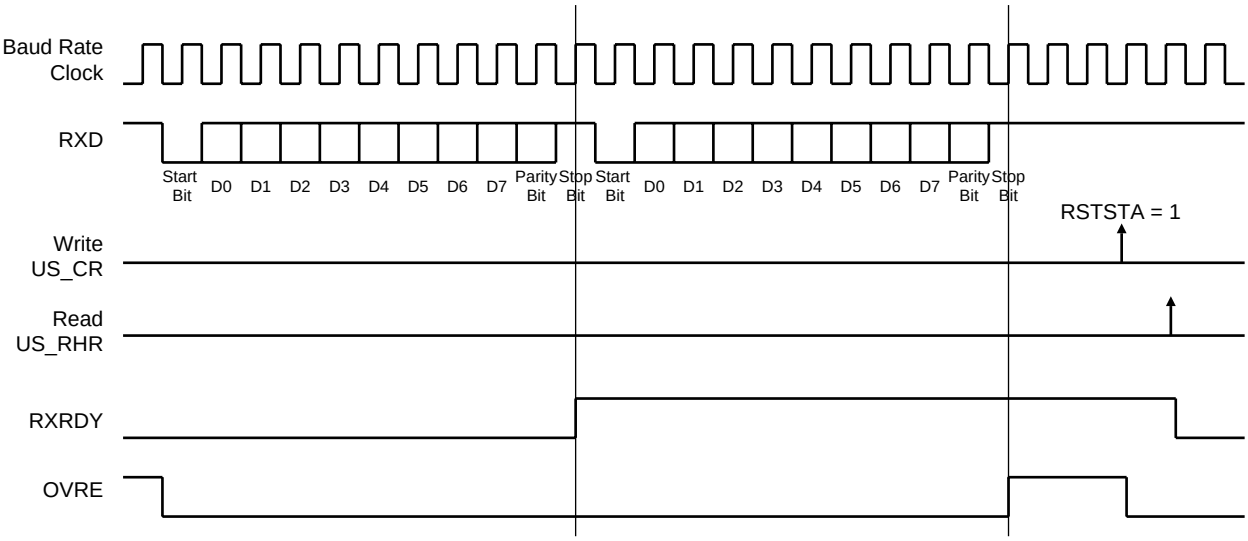
Figure 98. Synchronous Mode Character Reception



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 99. 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.

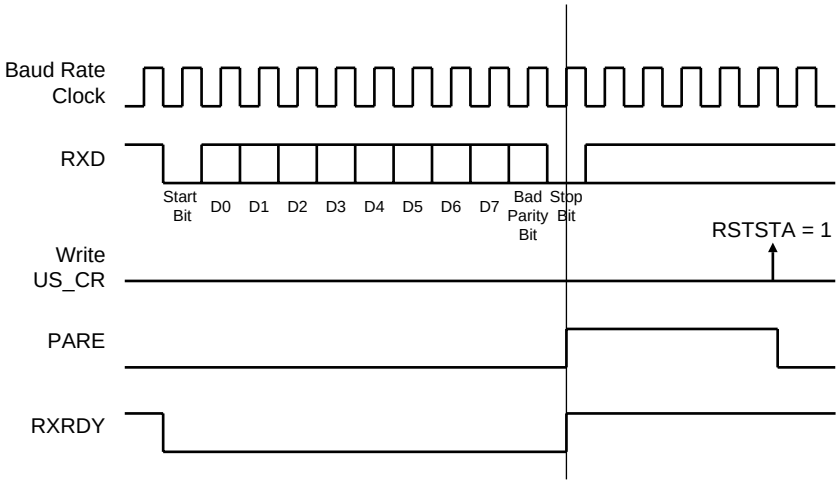
Table 48 shows an example of the parity bit for the character 0x41 (character ASCII “A”) depending on the configuration of the USART. Because there are two bits at 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even. I

Table 48. Parity Bit Examples

Character	Hexa	Binary	Parity Bit	ParityMode
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 100 illustrates the parity bit status setting and clearing.

Figure 100. Parity Error



Multi-drop Mode

If the PAR field in the Mode Register (US_MR) is programmed to the value 0x3, 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 101, 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 101. Timeguard Operations

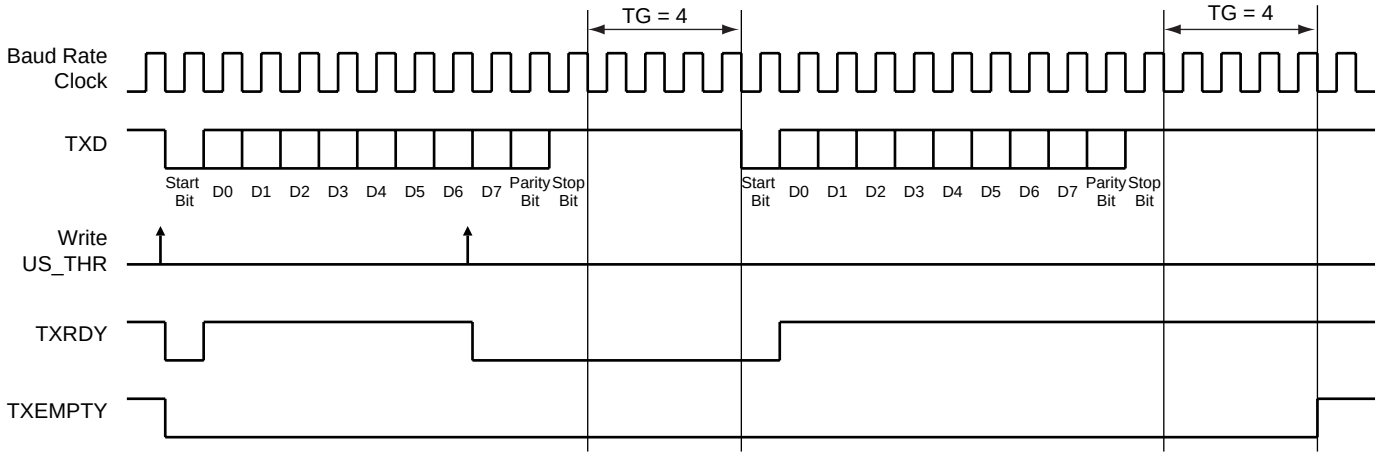


Table 49 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

Table 49. 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 102 shows the block diagram of the Receiver Time out feature.

Figure 102. Receiver Time-out Block Diagram

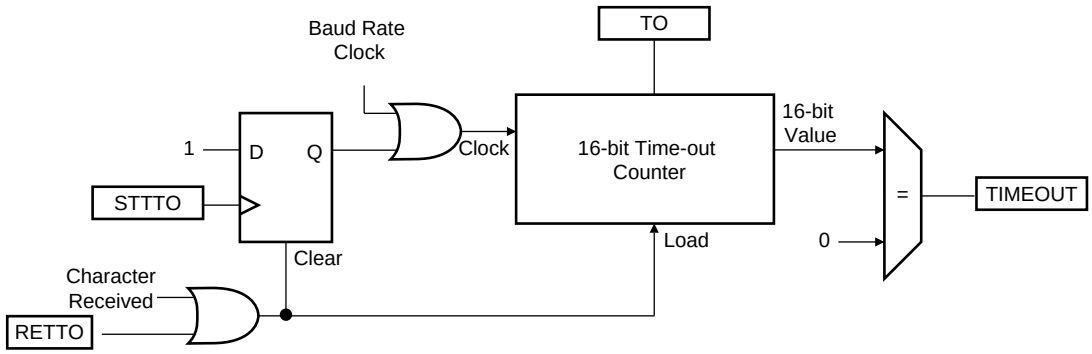


Table 50 gives the maximum time-out period for some standard baud rates.

Table 50. Maximum Time-out Period

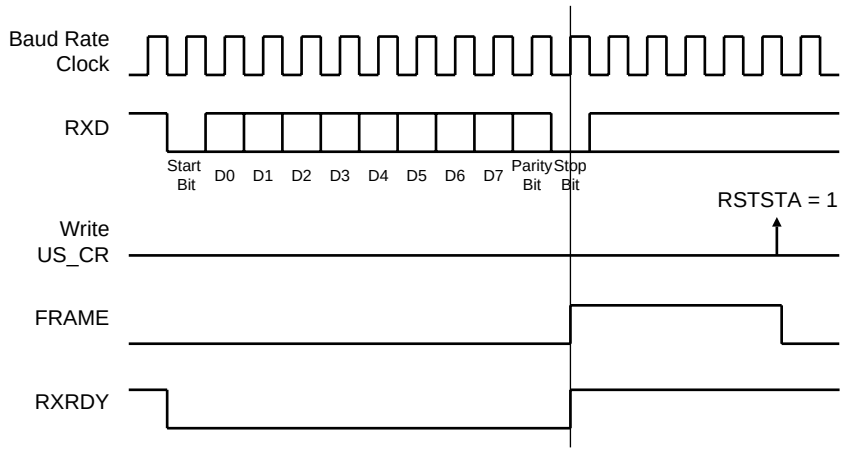
Baud Rate	Bit Time	Time -out
bit/sec	µs	ms
600	1 667	109 225
1 200	833	54 613
2 400	417	27 306
4 800	208	13 653
9 600	104	6 827
14400	69	4 551
19200	52	3 413
28800	35	2 276
33400	30	1 962
56000	18	1 170
57600	17	1 138
200000	5	328

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 103. 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 STTBRK 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 STTBRK command is requested further STTBRK commands are ignored until the end of the break is completed.

The break condition is removed by writing US_CR with the STPBRK bit 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 STTBRK 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.

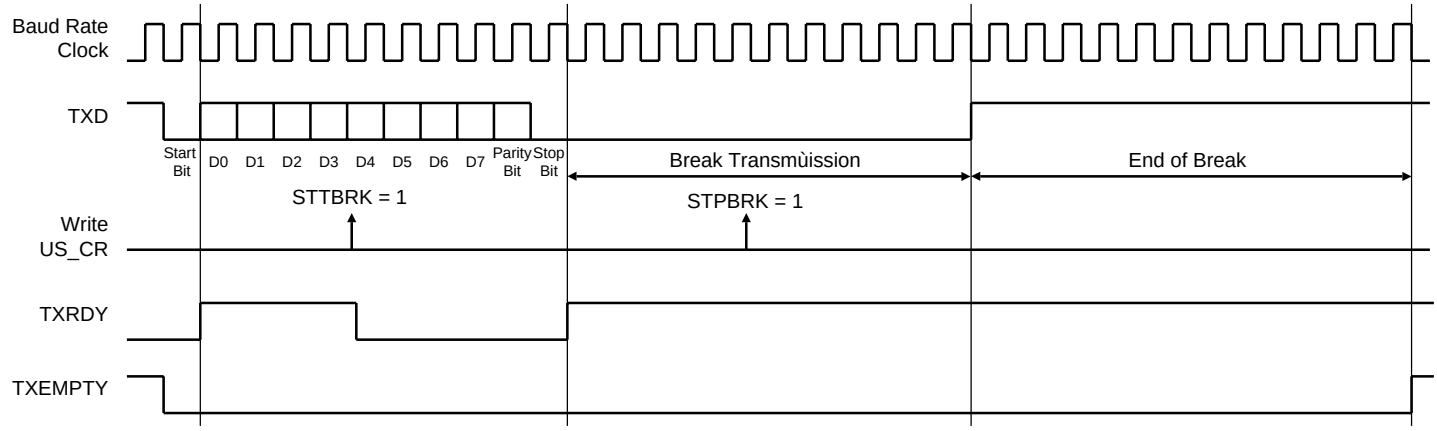
Writing US_CR with the both STTBRK and STPBRK bits at 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

After the break condition, the transmitter returns the TXD line to 1 for a minimum of 12 bit times. Thus, the transmitter ensures that the remote receiver detects correctly the end of break and the start of the next character. If the timeguard is programmed with a value higher than 12, the TXD line is held high for the timeguard period.

After holding the TXD line for this period, the transmitter resumes normal operations.

Figure 104 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STP BRK) commands on the TXD line.

Figure 104. 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.

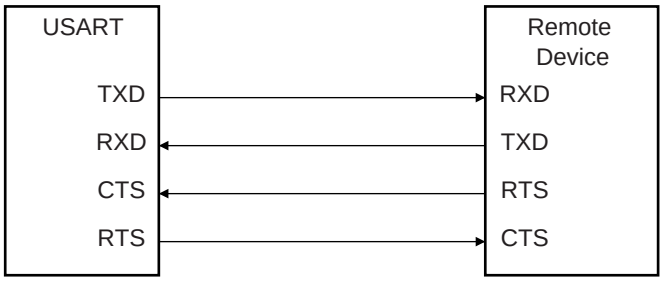
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing the Control Register (US_CR) with the bit RSTSTA at 1.

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 105.

Figure 105. 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 106 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 106. Receiver Behavior when Operating with Hardware Handshaking

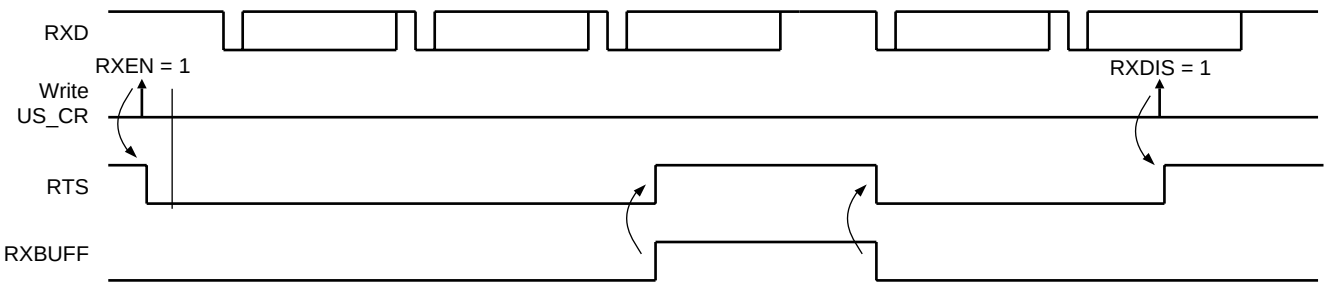
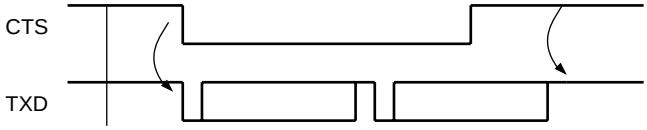


Figure 107 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

Figure 107. 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.

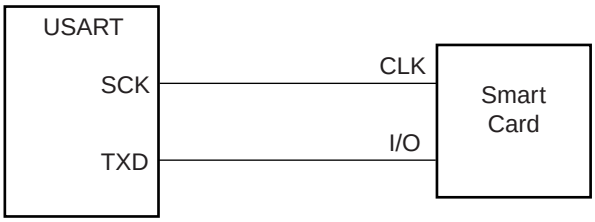
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in the Mode Register (US_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

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 271).

The USART connects to a smart card. as shown in Figure 108. 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 108. 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 109.

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 110. 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 109. T = 0 Protocol without Parity Error

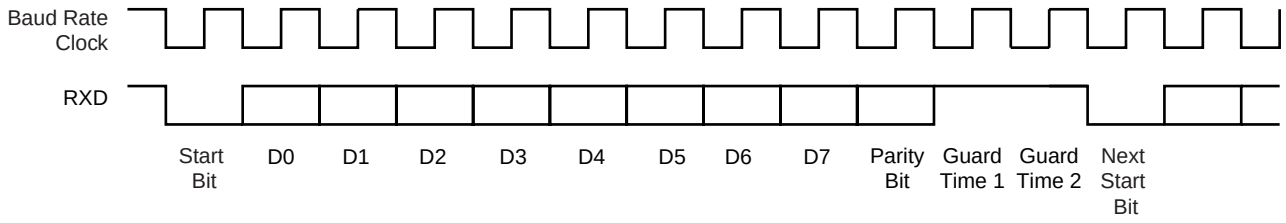
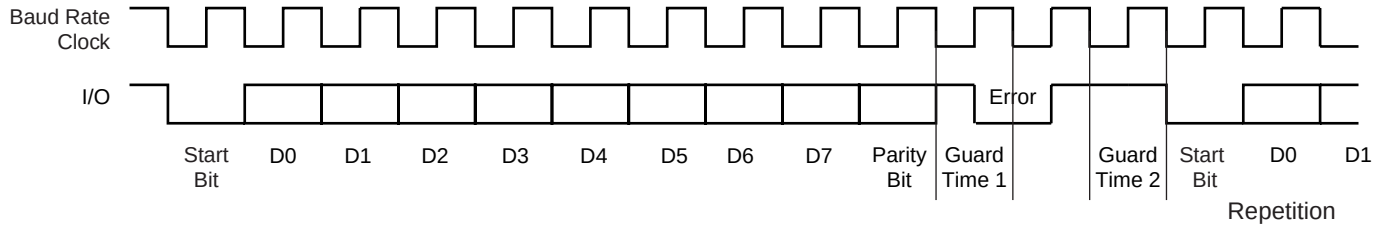


Figure 110. T = 0 Protocol with Parity Error



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

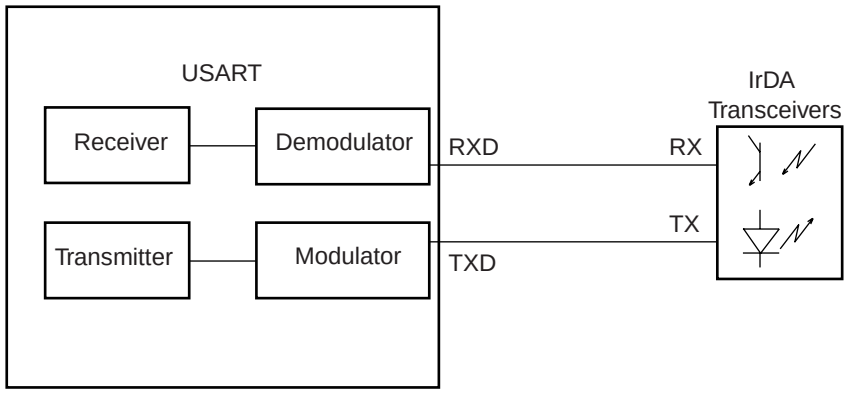
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US_CSR).

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 111. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2,4 Kbps to 115,2 Kbps.

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 111. 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

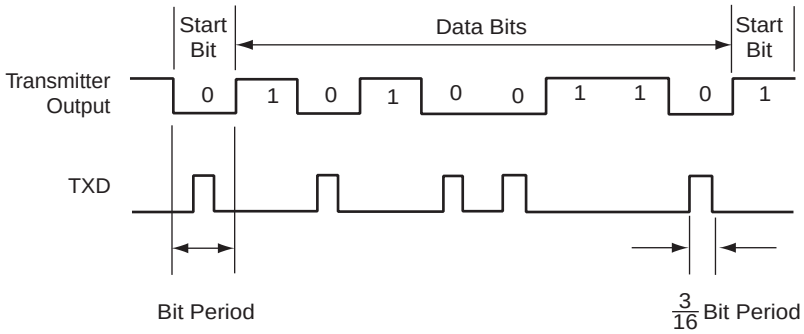
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. "0" is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 51..

Table 51. IrDA Pulse Duration

Baud Rate	Pulse Duration (3/16)
2.4 Kb/s	78.13 µs
9.6 Kb/s	19.53 µs
19.2 Kb/s	9.77 µs
38.4 Kb/s	4.88 µs
57.6 Kb/s	3.26 µs
115.2 Kb/s	1.63 µs

Figure 112 shows an example of character transmission.

Figure 112. IrDA Modulation



IrDA Baud Rate

Table 52 gives some examples of CD values, baud rate error and pulse duration. Note that the requirement on the maximum acceptable error of +/- 1.87% must be met.

Table 52. IrDA Baud Rate Error

Peripheral Clock	Baud rate	CD	Baud rate Error	Pulse time
3 686 400	115 200	2	0.00%	1.63
20 000 000	115 200	11	1.38%	1.63
32 768 000	115 200	18	1.25%	1.63
40 000 000	115 200	22	1.38%	1.63
3 686 400	57 600	4	0.00%	3.26
20 000 000	57 600	22	1.38%	3.26
32 768 000	57 600	36	1.25%	3.26
40 000 000	57 600	43	0.93%	3.26
3 686 400	38 400	6	0.00%	4.88
20 000 000	38 400	33	1.38%	4.88
32 768 000	38 400	53	0.63%	4.88

Table 52. IrDA Baud Rate Error

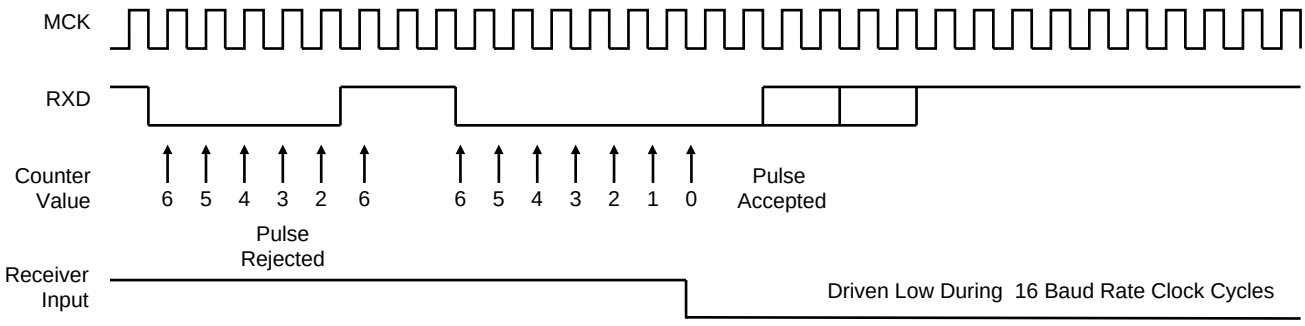
Peripheral Clock	Baud rate	CD	Baud rate Error	Pulse time
40 000 000	38 400	65	0.16%	4.88
3 686 400	19 200	12	0.00%	9.77
20 000 000	19 200	65	0.16%	9.77
32 768 000	19 200	107	0.31%	9.77
40 000 000	19 200	130	0.16%	9.77
3 686 400	9 600	24	0.00%	19.53
20 000 000	9 600	130	0.16%	19.53
32 768 000	9 600	213	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 113 illustrates the operations of the IrDA demodulator.

Figure 113. IrDA Demodulator Operations

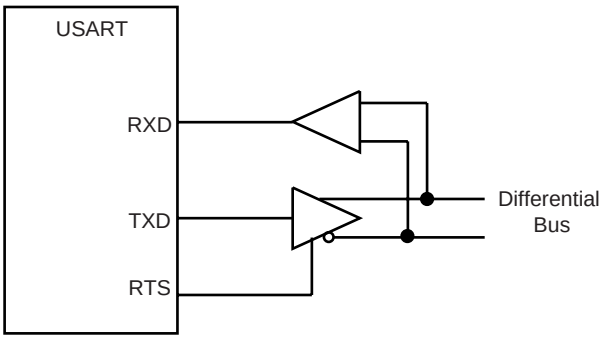


As the IrDA mode uses the same logic as the ISO7816, note that the FI_DI_RATIO field in US_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

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 are 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 114.

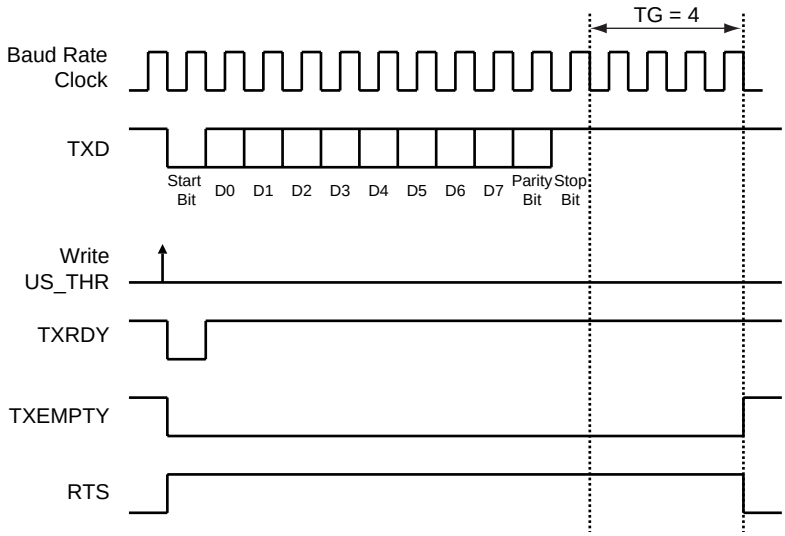
Figure 114. Typical Connection to a RS485 bus.



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 of 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 115 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

Figure 115. 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 53 gives the correspondence of the USART signals with modem connection standards.

Table 53. 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 witting 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.

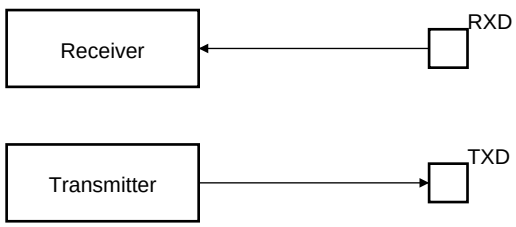
Test Modes

The USART can be programmed to operate in three different test modes. The internal loopback capability allows on-board diagnostics. In the loopback mode the USART interface pins are disconnected or not and reconfigured for loopback internally or externally.

Normal Mode

As a reminder, the normal mode simply connects the RXD pin on the receiver input and the transmitter output on the TXD pin.

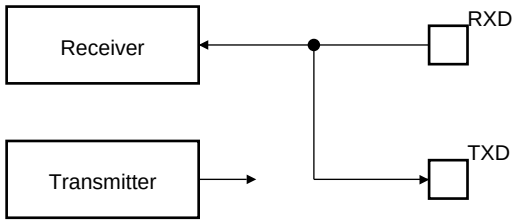
Figure 116. Normal Mode Configuration



Automatic Echo

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 117. 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.

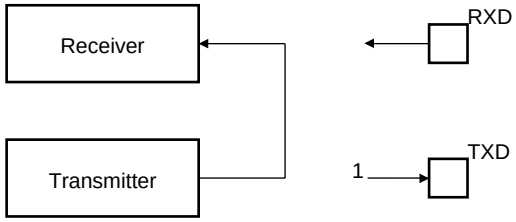
Figure 117. Automatic Echo



Local Loopback

The local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 118. 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.

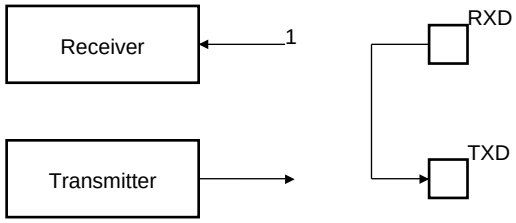
Figure 118. Local Loopback



Remote Loopback

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 119. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 119. Remote Loopback



USART User Interface

Table 54. USART Memory Map

Offset	Register	Name	Access	Reset State
0x0000	Control Register	US_CR	Write-only	–
0x0004	Mode Register	US_MR	Read/Write	–
0x0008	Interrupt Enable Register	US_IER	Write-only	–
0x000C	Interrupt Disable Register	US_IDR	Write-only	–
0x0010	Interrupt Mask Register	US_IMR	Read-only	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
0x2C to 0x3C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read/Write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	–
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read/Write	0
0x5C to 0xFC	Reserved	–	–	–
0x100 to 0x128	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	IS07816 Protocol: T = 0
0	1	0	1	Reserved
0	1	1	0	IS07816 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.

Serial Synchronous 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:

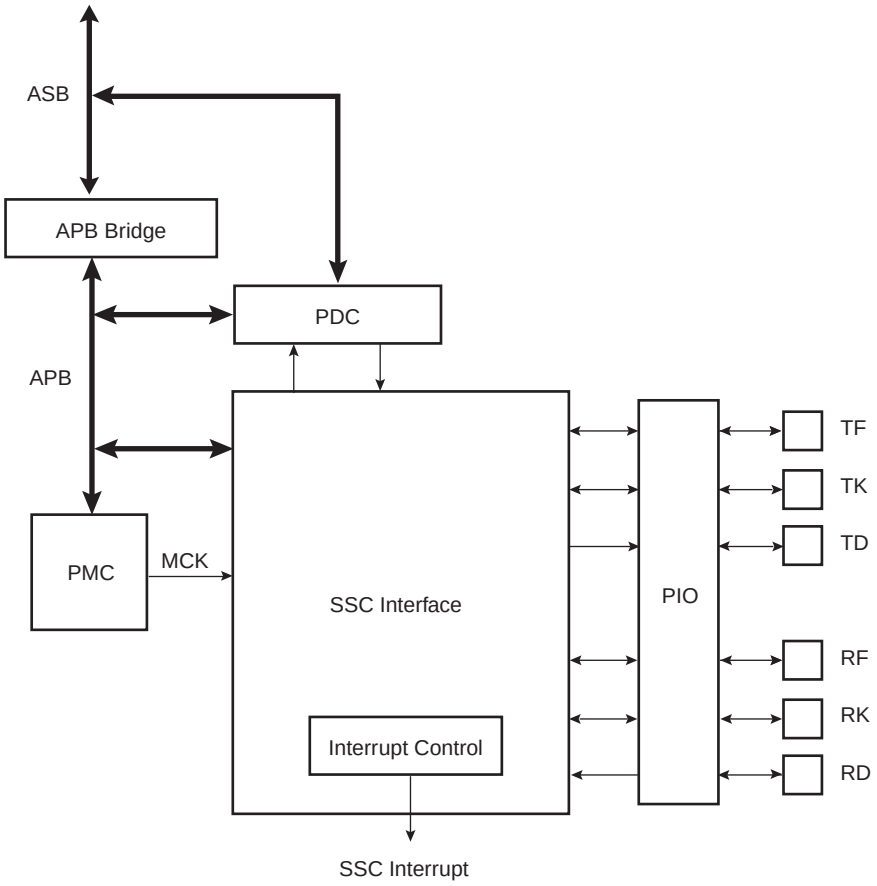
- CODECs in master or slave mode
- DAC through dedicated serial interface, particularly I2S
- Magnetic card reader

Features of the SSC are:

- Provides Serial Synchronous Communication Links Used in Audio and Telecom Applications
- Contains an Independent Receiver and Transmitter and a Common Clock Divider
- Interfaced with Two PDC Channels (DMA Access) to Reduce Processor Overhead
- 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

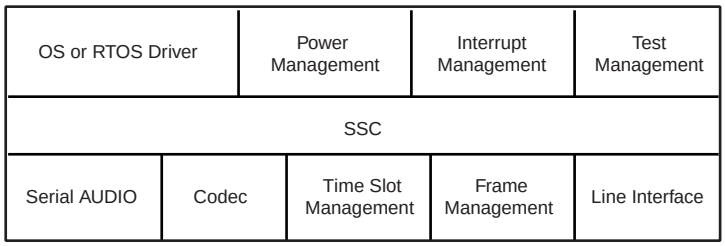
Block Diagram

Figure 120. Block Diagram



Application Block Diagram

Figure 121. Application Block Diagram



Pin Name List

Table 55. 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.

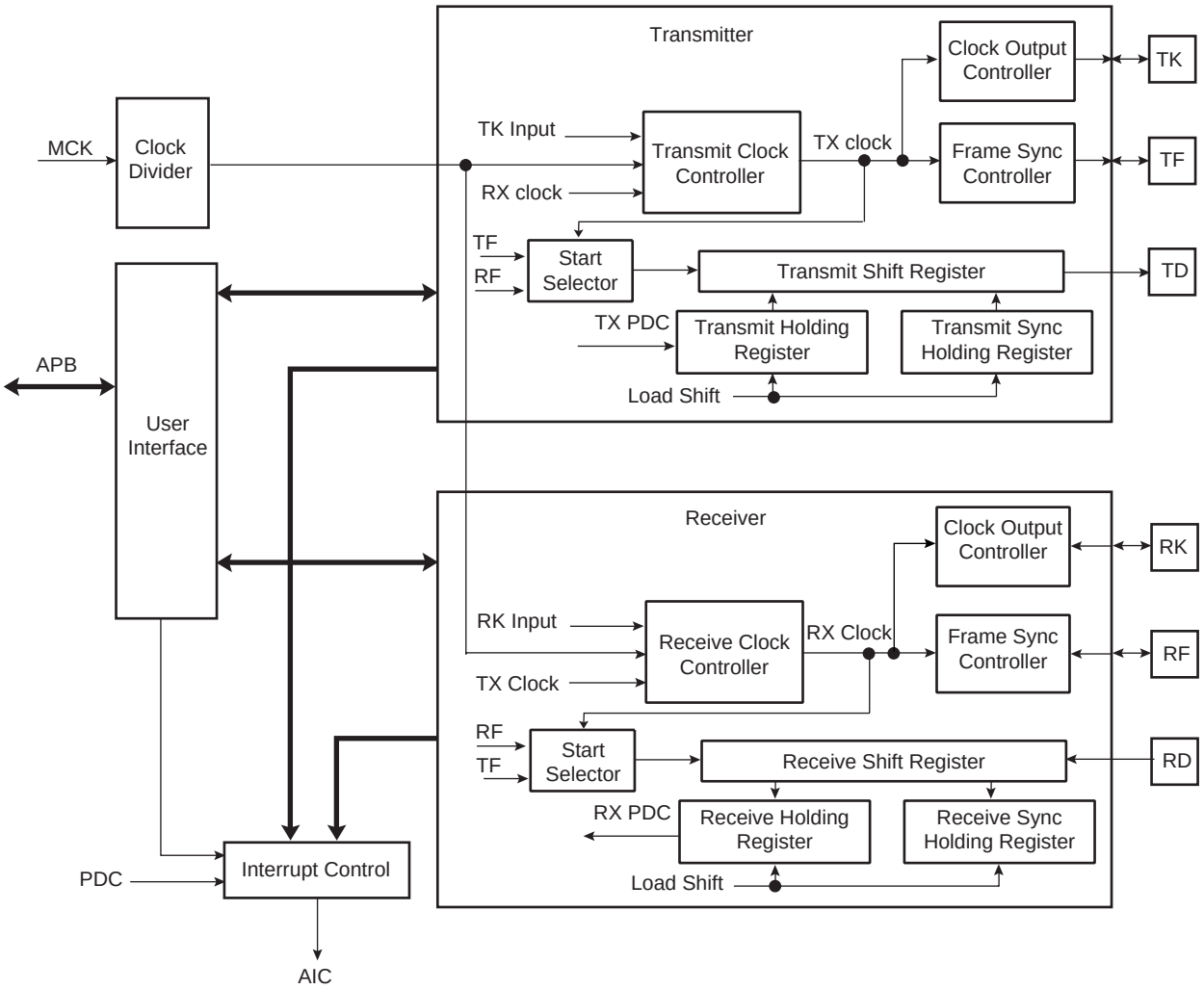
All SSC interrupts can be enabled/disabled configuring the SSC Interrupt mask register. Each pending and unmasked SSC interrupt will assert the SSC interrupt line. The SSC interrupt service routine can get the interrupt origin by reading the SSC interrupt status register.

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 122. 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:

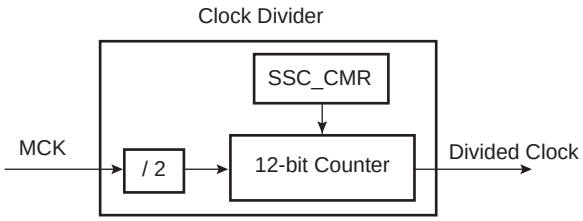
- an external clock received on the RK I/O pad
- the transmitter clock
- the internal clock divider

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 123. 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_CMR, 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 124. Divided Clock Generation

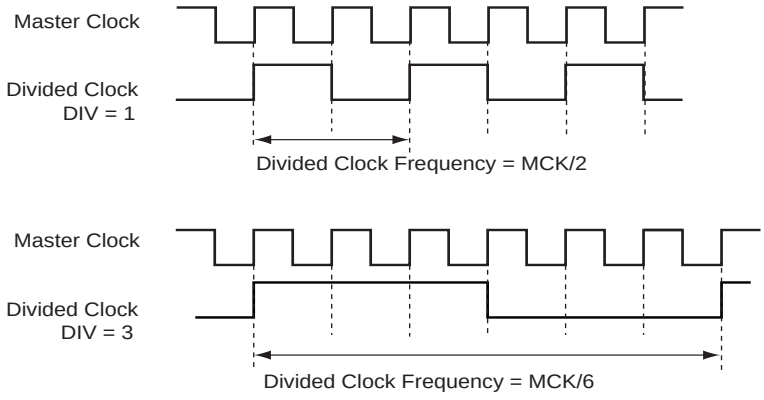


Table 56. Bit Rate

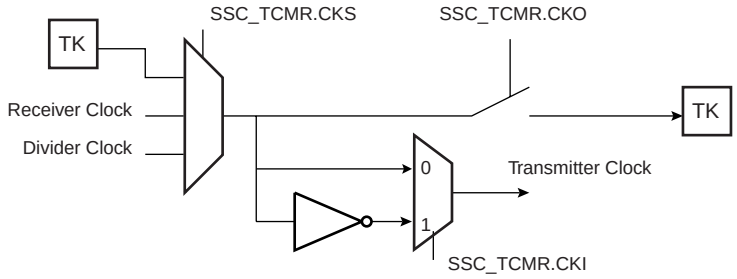
Maximum	Minimum
MCK / 2	MCK / 8190

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 125. Transmitter Clock Management

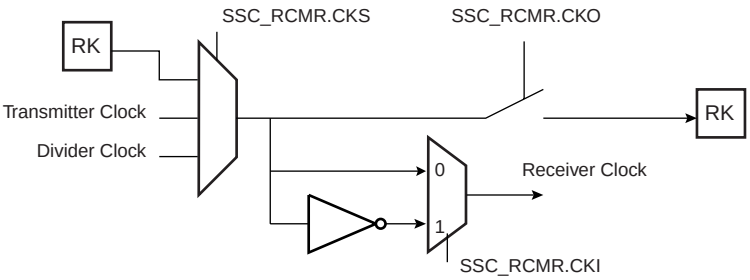


Receiver Clock Management

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 126. Receiver Clock Management



Transmitter Operations

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

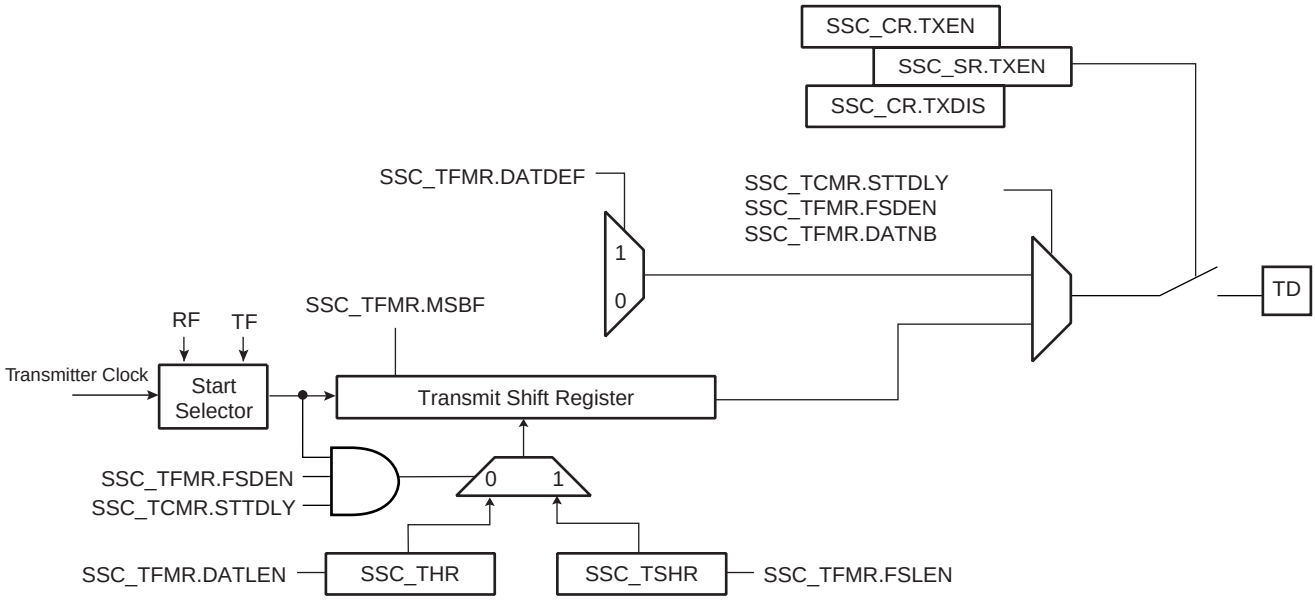
The start event is configured by setting the Transmit Clock Mode Register (SSC_TCMR). See “Start” on page 318.

The frame synchronization is configured setting the Transmit Frame Mode Register (SSC_TFMR). See “Frame Sync” on page 320.

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 127. Transmitter Block Diagram



Receiver Operations

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

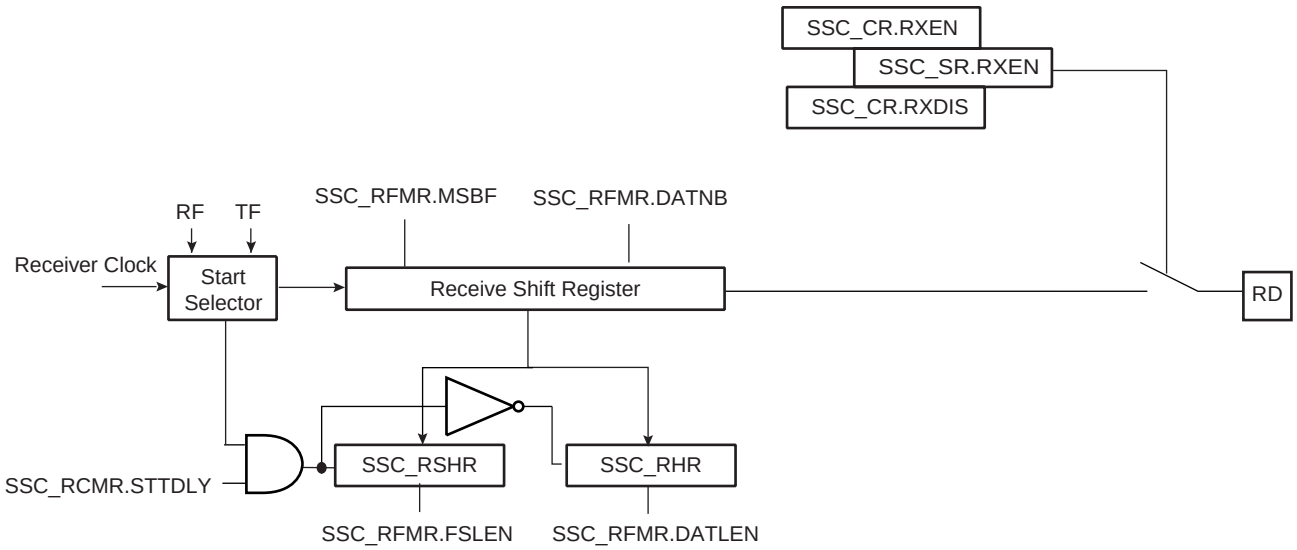
The start event is configured setting the Receive Clock Mode Register (SSC_RCMR). See “Start” on page 318.

The frame synchronization is configured setting the Receive Frame Mode Register (SSC_RFMR). See “Frame Sync” on page 320.

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 128. 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 129. Transmit Start Mode

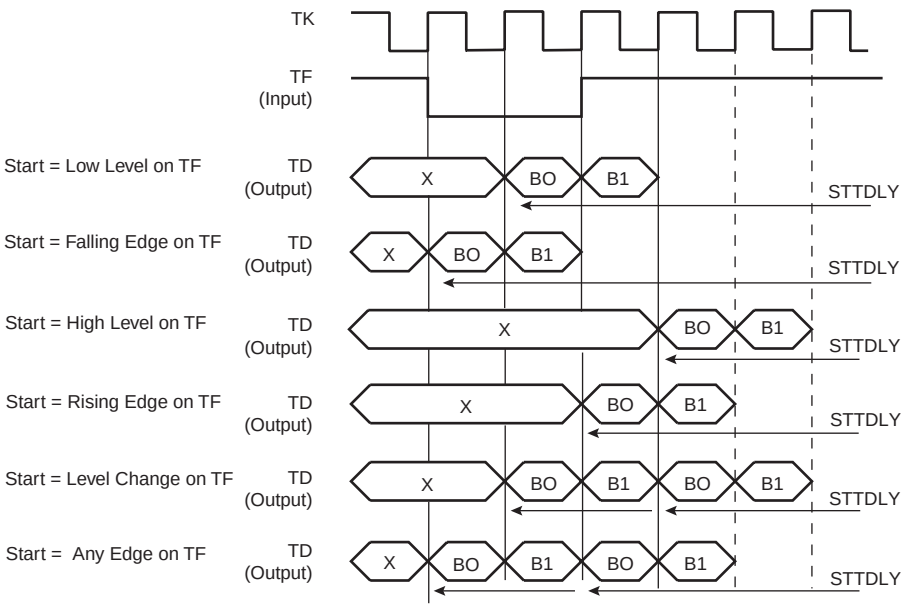
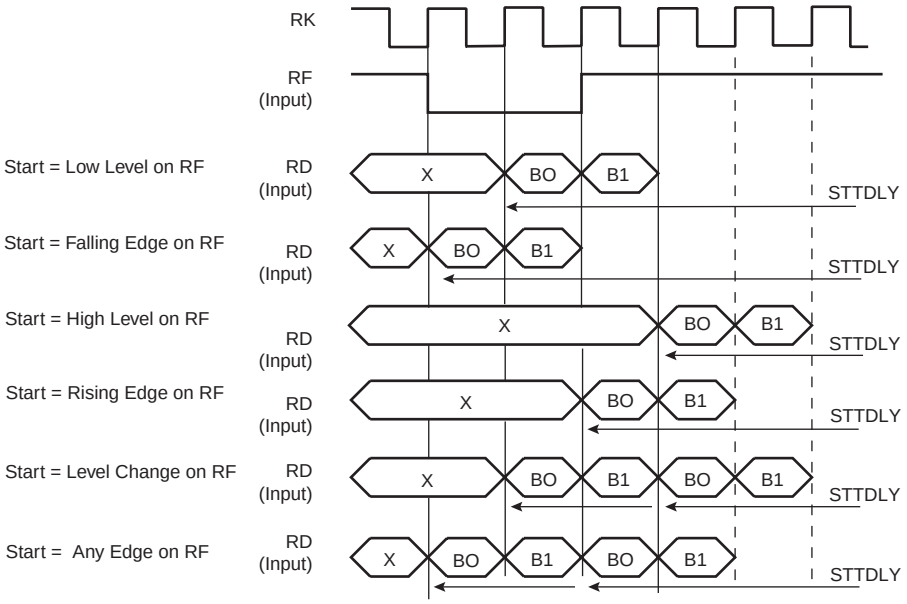


Figure 130. Receive Pulse/Edge Start Modes



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:

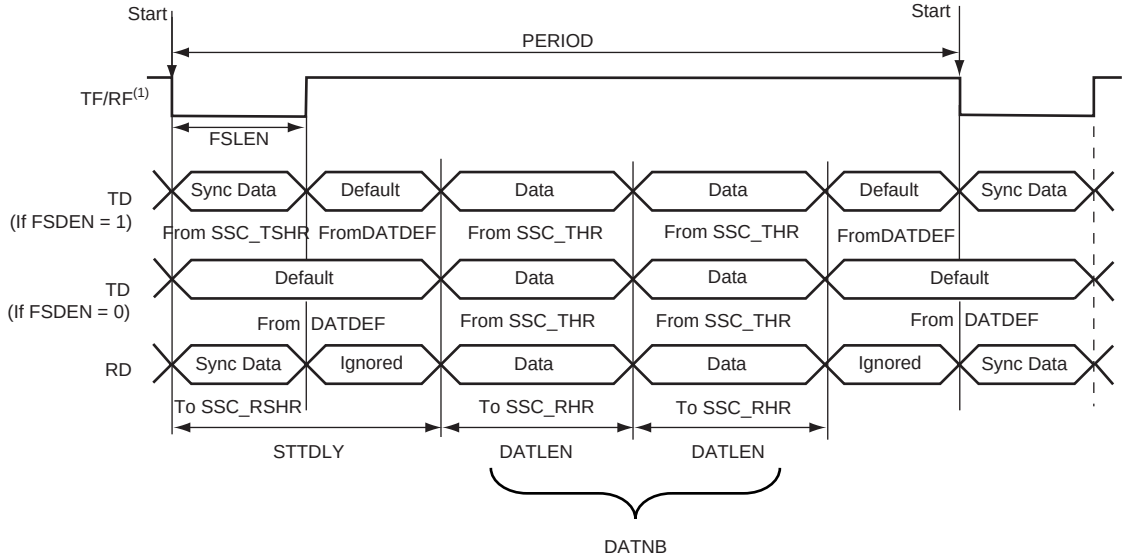
- The event that starts the data transfer (START).
- The delay in number of bit periods between the start event and the first data bit (STTDLY).
- The length of the data (DATLEN)
- The number of data to be transferred for each start event (DATNB).
- The length of Synchronization transferred for each 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 57. Data Frame Registers

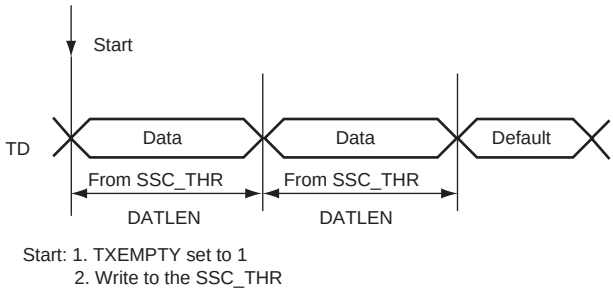
Transmitter	Receiver	Field	Length	Comment
SSC_TFMR	SSC_RFMR	DATLEN	Up to 32	Size of word
SSC_TFMR	SSC_RFMR	DATNB	Up to 16	Number Word transmitter in frame
SSC_TFMR	SSC_RFMR	MSBF		1 most significant bit in first
SSC_TFMR	SSC_RFMR	FSLEN	Up to 16	Size of Synchro data register
SSC_TFMR		DATDEF	0 or 1	Data 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 131. Transmit and Receive Frame Format in Edge/Pulse Start Modes



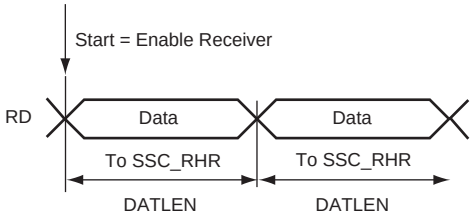
Note: 1. Input on falling edge on TF/RF example.

Figure 132. Transmit Frame Format in Continuous Mode



Note: 1. STTDLY is set to 0. In this example, SSC_THR is loaded twice. The value of FSDEN has no effect on transmission. SyncData cannot be output in continuous mode.

Figure 133. Receive Frame Format in Continuous Mode



Note: 1. STTDLY is set to 0.

Loop Mode

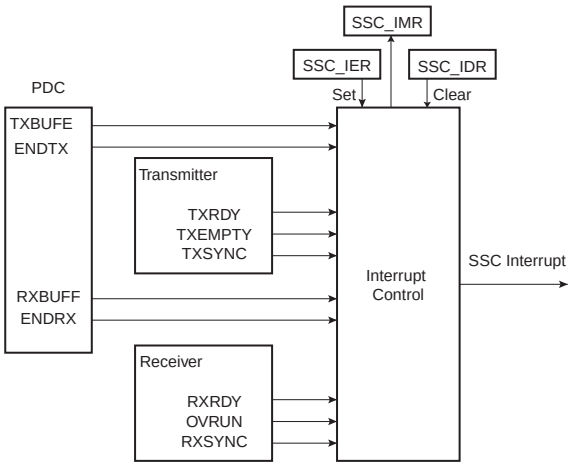
The receiver can be programmed to receive transmissions from the transmitter. This is done by setting the Loop Mode (LOOP) bit in SSC_RFMR. In this case, RD is connected to TD, RF is connected to TF and RK is connected to TK.

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 134. 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 135. Audio Application Block Diagram

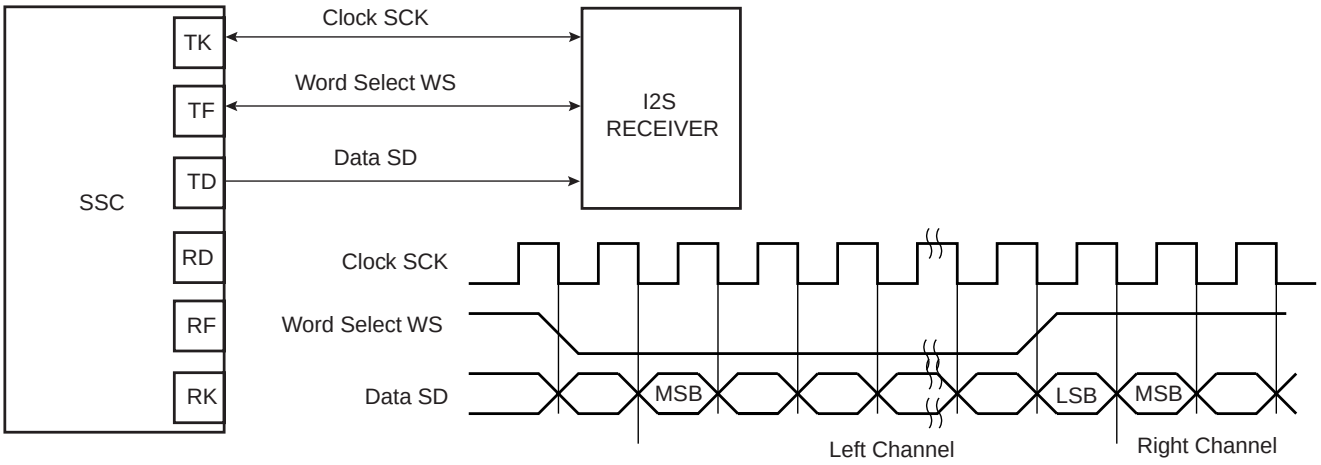


Figure 136. Codec Application Block Diagram

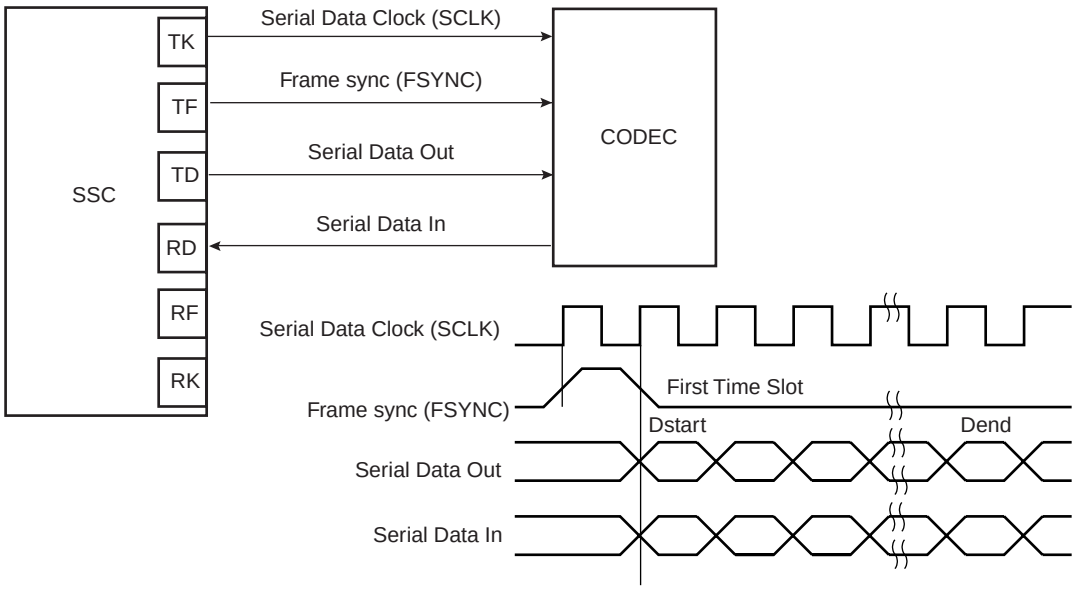
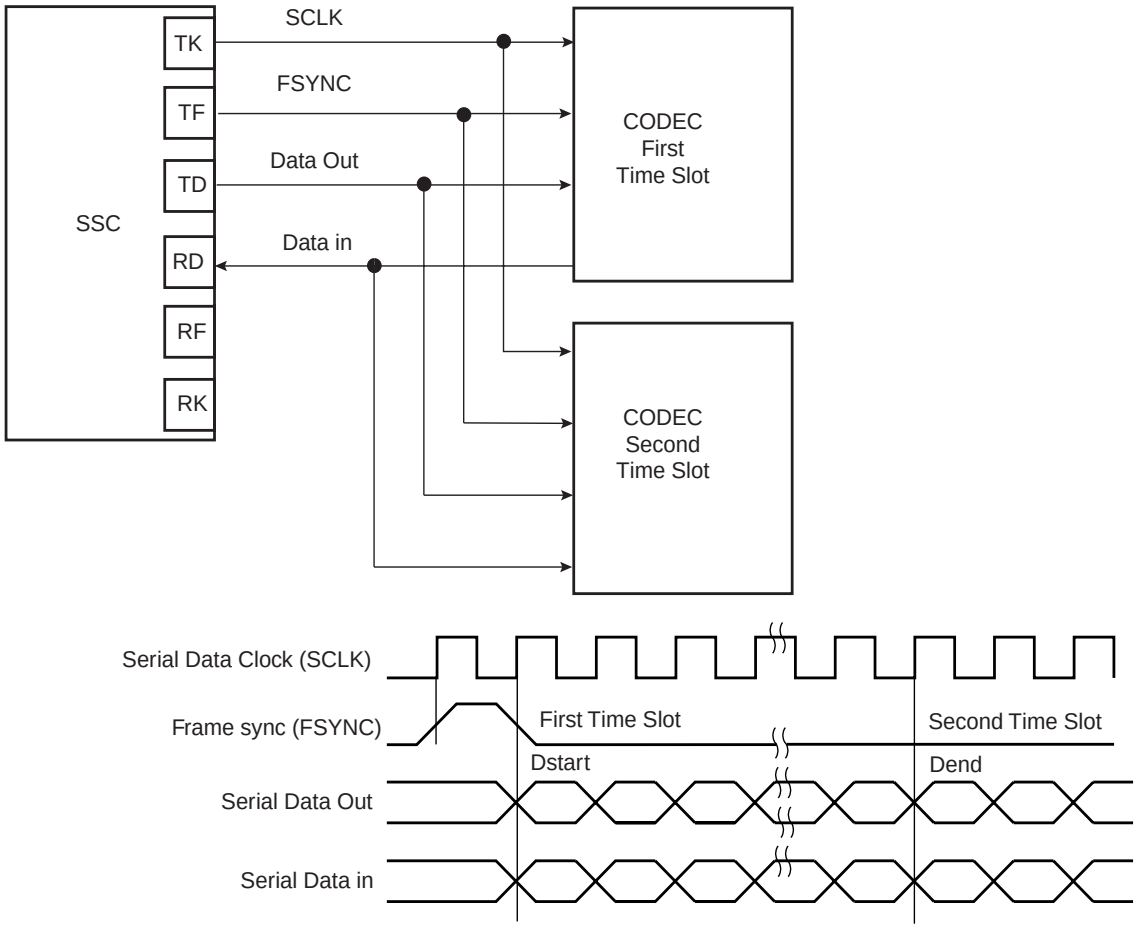


Figure 137. Time Slot Application Block Diagram



Serial Synchronous Controller (SSC) User Interface

Table 58. 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	–	–	–

Table 58. SSC Register Mapping

Offset	Register	Register Name	Access	Reset
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
0x50-0xFF	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(1).

- **RXDIS: Receive Disable**

0: No effect.

1: Disables Data Receive(1).

- **TXEN: Transmit Enable**

0: No effect.

1: Enables Data Transmit if TXDIS is not set(1).

- **TXDIS: Transmit Disable**

0: No effect.

1: Disables Data Transmit(1).

- **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 affects 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 x (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.

FSEDGE	Frame Sync Edge Detection
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 with 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 x (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).

FSEDGE	Frame Sync Edge Detection
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.

Key Features of the Timer Counter are:

- Three 16-bit Timer Counter Channels
- A 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
  - Two Multi-purpose Input/Output Signals
- Internal Interrupt Signal

Two Global Registers that Act on All Three TC Channels

Block Diagram

Figure 138. Timer Counter Block Diagram

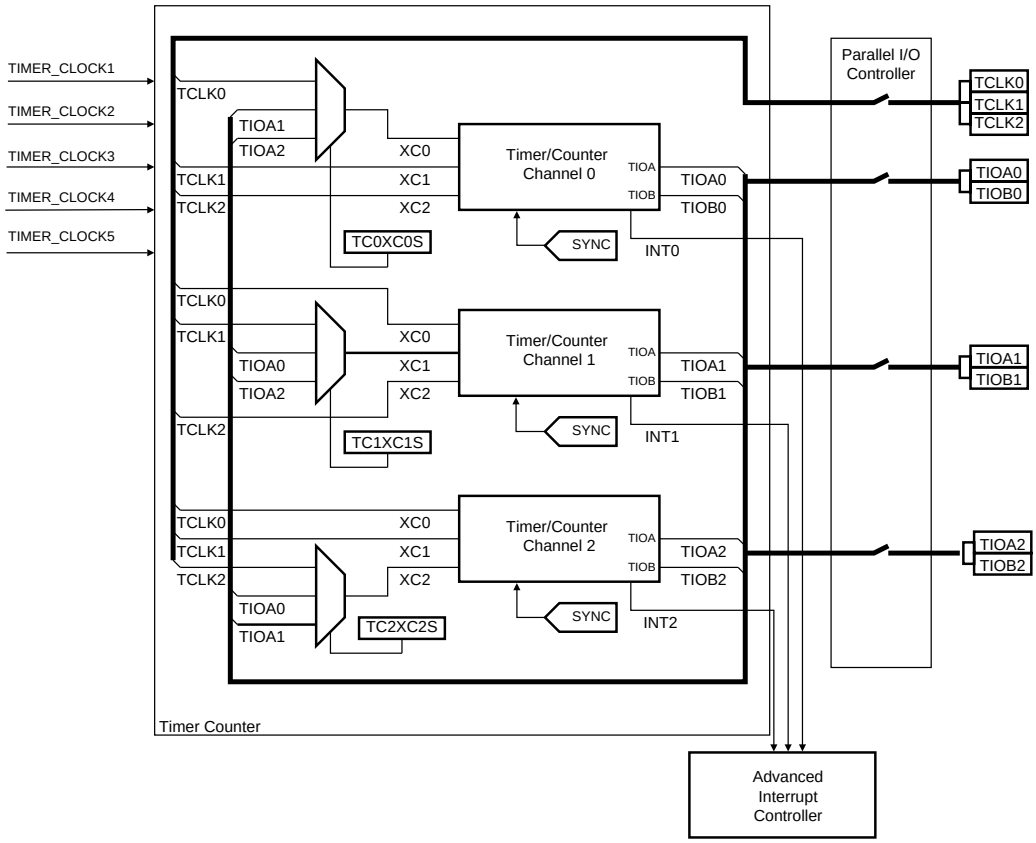


Table 59. Signal Name Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: General-purpose Input Waveform Mode: General-purpose Output
	TIOB	Capture Mode: General-purpose Input Waveform Mode: General-purpose Input/output
	INT[2:0]	TC Internal Interrupt Signal Output
	SYNC	Synchronization Input Signal
Block Signal	TCLK0, TCLK1, TCLK2	External Clock Inputs
	TIOA0	TIOA Signal for Channel 0
	TIOB0	TIOB Signal for Channel 0
	TIOA1	TIOA Signal for Channel 1
	TIOB1	TIOB Signal for Channel 1
	TIOA2	TIOA Signal for Channel 2
	TIOB2	TIOB Signal for Channel 2

Pin Name List

Table 60. Timer Counter pin list

Pin Name	Description	Type
TCLK0-TCLK2	External Clock Input	Input
TIOA0-TIOA2	I/O Line A	I/O
TIOB0-TIOB2	I/O Line B	I/O

Product Dependencies

For further details on the Timer Counter hardware implementation, see the specific Product Properties document.

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 must be clocked through the Power Management Controller (PMC), thus the programmer must first configure the PMC to enable the Timer Counter.

Interrupt

The TC interface 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 60 on page 345.

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, TIOA1 or TIOA2 for chaining by programming the TC_BMR (Block Mode). See Figure 139.

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

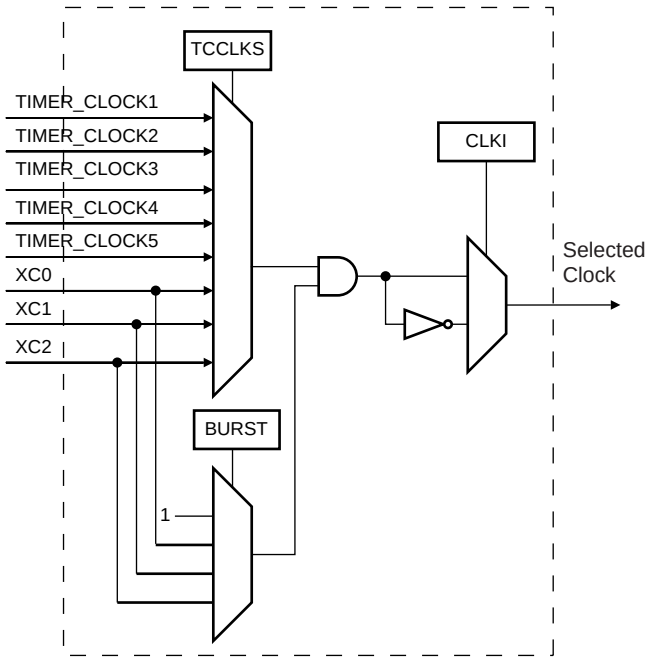
This selection is made by the TCCLKS bits in the TC Channel Mode Register (Capture Mode).

The selected clock can be inverted with the CLKI bit in TC_CMR (Capture Mode). 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 139. Clock Selection

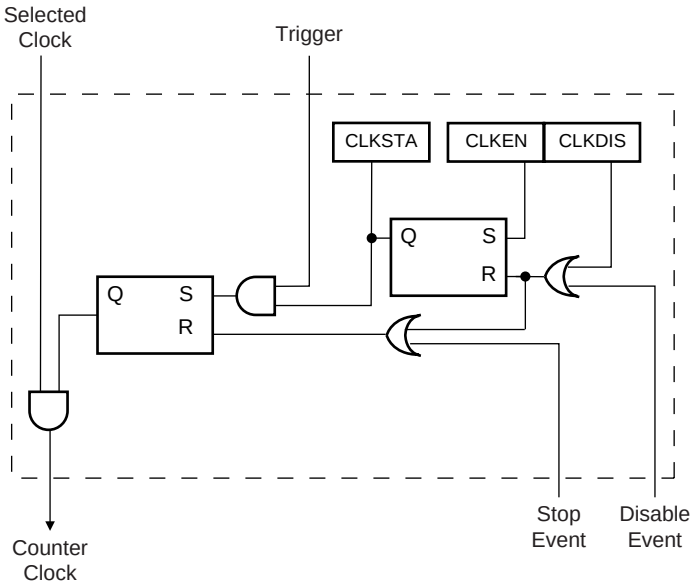


Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 140.

- 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 140. 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 CPCTRG is set in TC_CMR.

The channel can also be configured to have an external trigger. In Capture Mode, the external trigger signal can be selected between TIOA and TIOB. In Waveform Mode, an external event can be programmed on one of the following signals: TIOB, XC0, XC1 or XC2. This external event can then be programmed to perform a trigger by setting ENETRG 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 141 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.

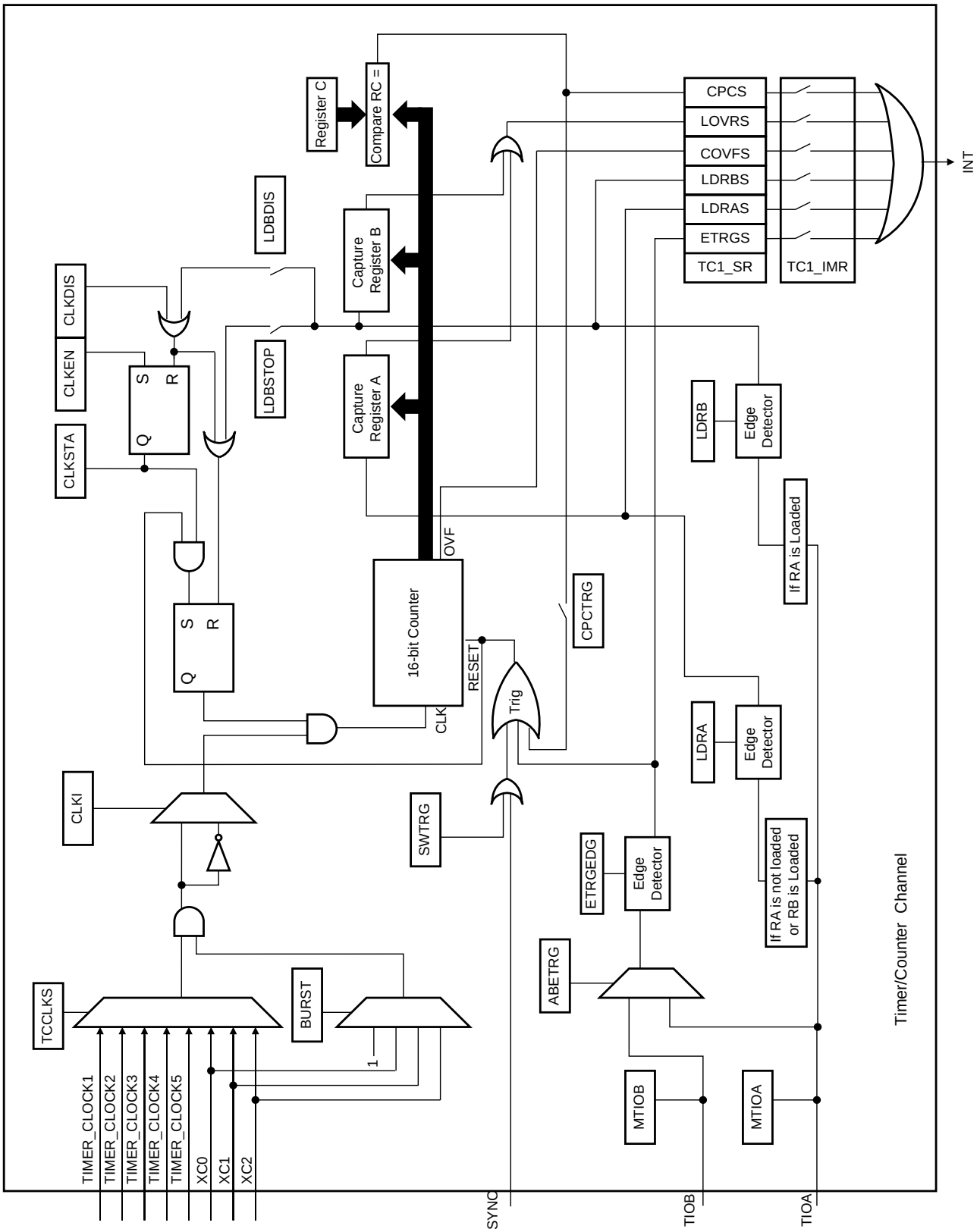
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS) in TC_SR (Status Register). In this case, the old value is overwritten.

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 141. 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 142 shows the configuration of the TC channel when programmed in Waveform Operating Mode.

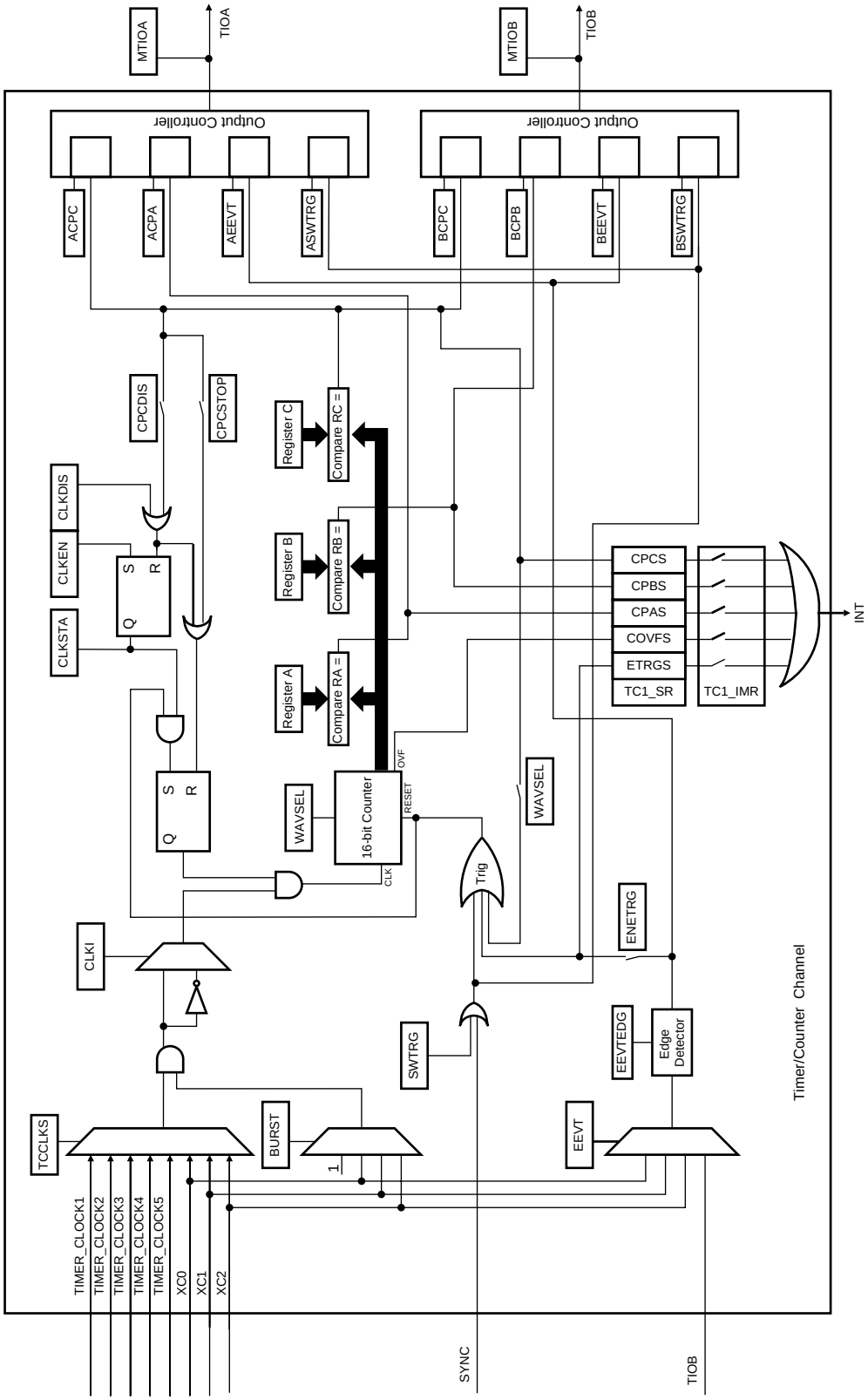
Waveform Selection

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 142. 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 143.

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 144.

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 143. WAVSEL= 00 without trigger

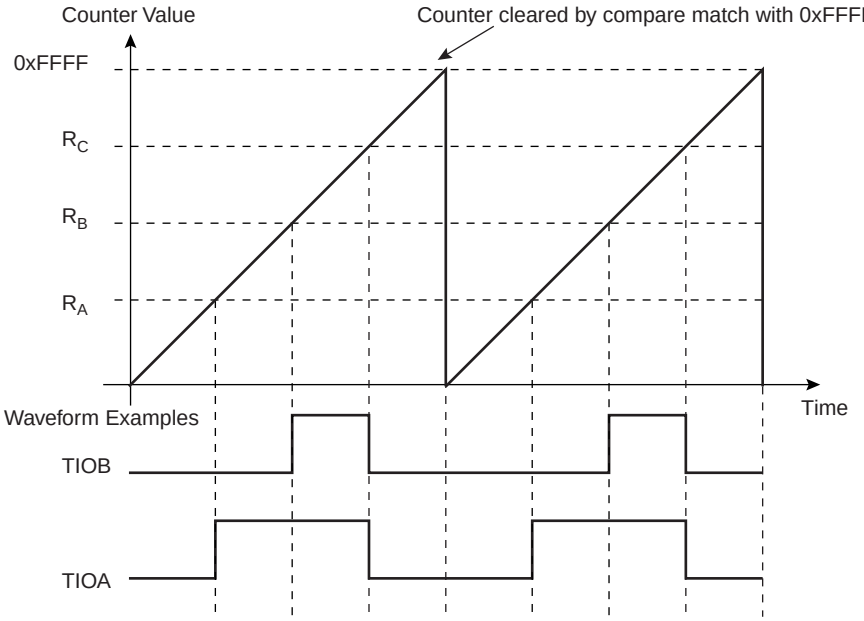
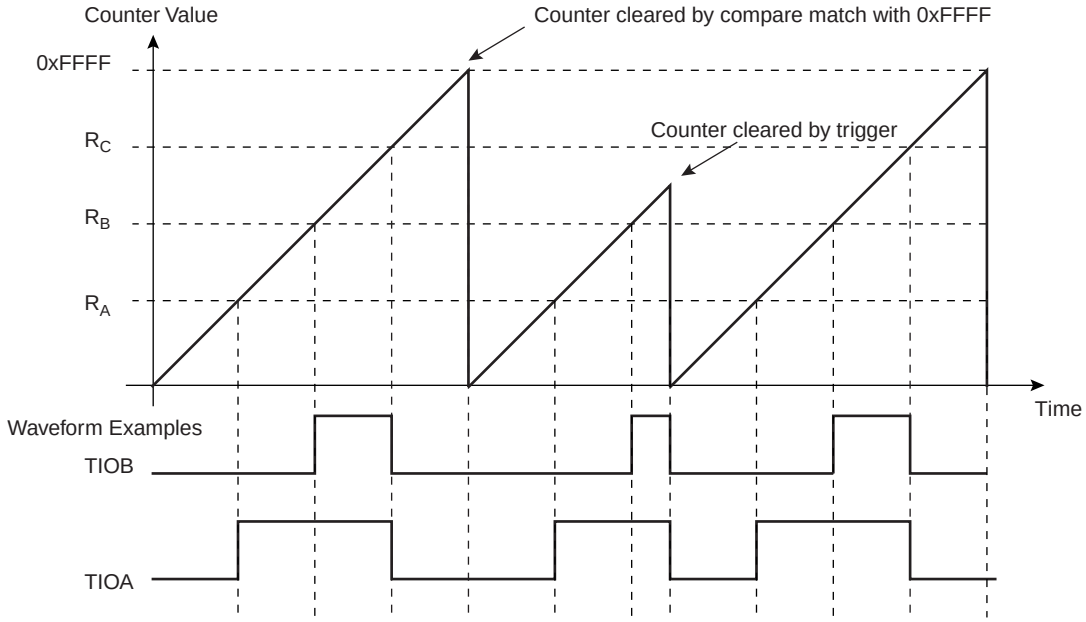


Figure 144. WAVSEL= 00 with trigger



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 145.

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 146.

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 145. WAVSEL = 10 Without Trigger

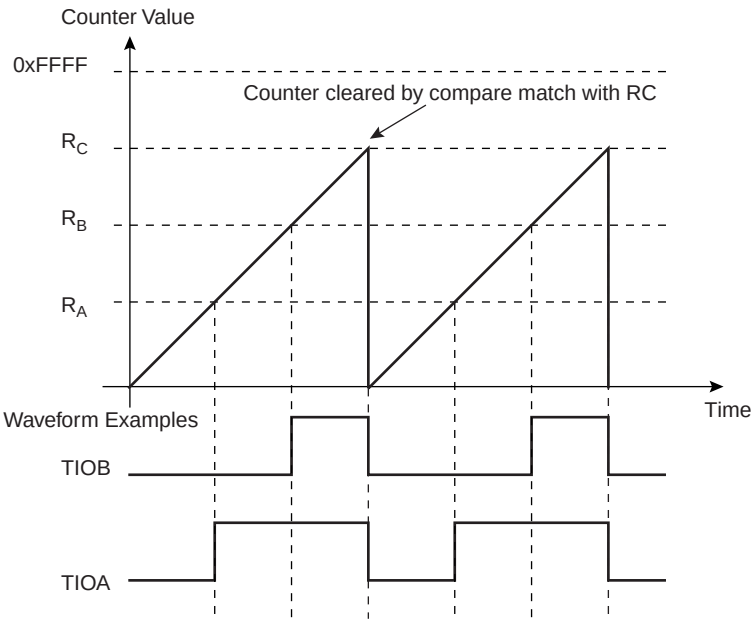
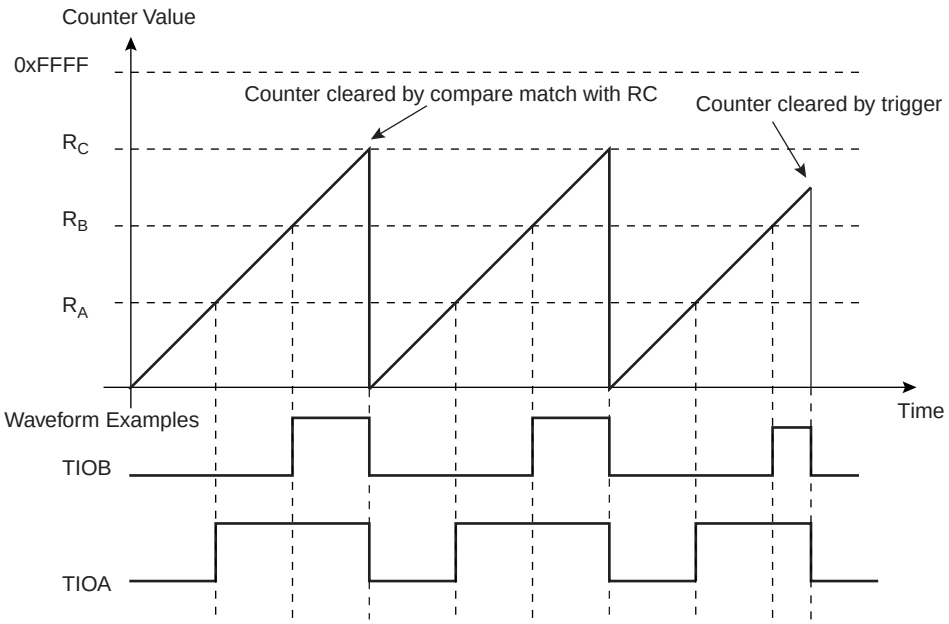


Figure 146. WAVSEL = 10 With Trigger



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 147.

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 148.

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 147. WAVSEL = 01 Without Trigger

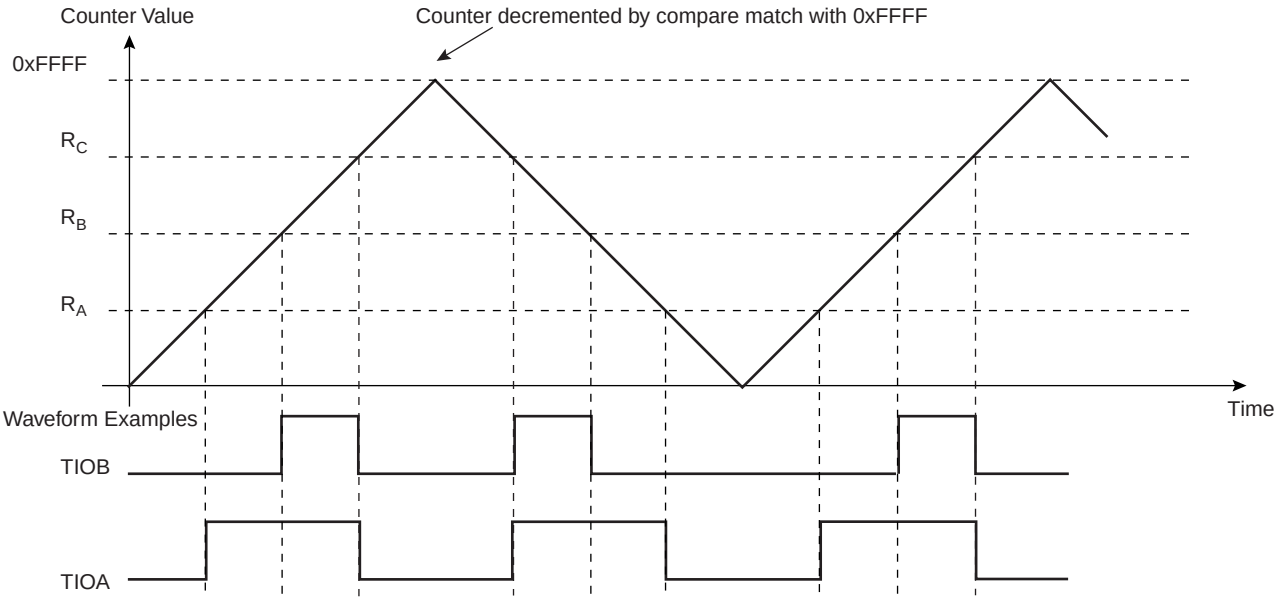
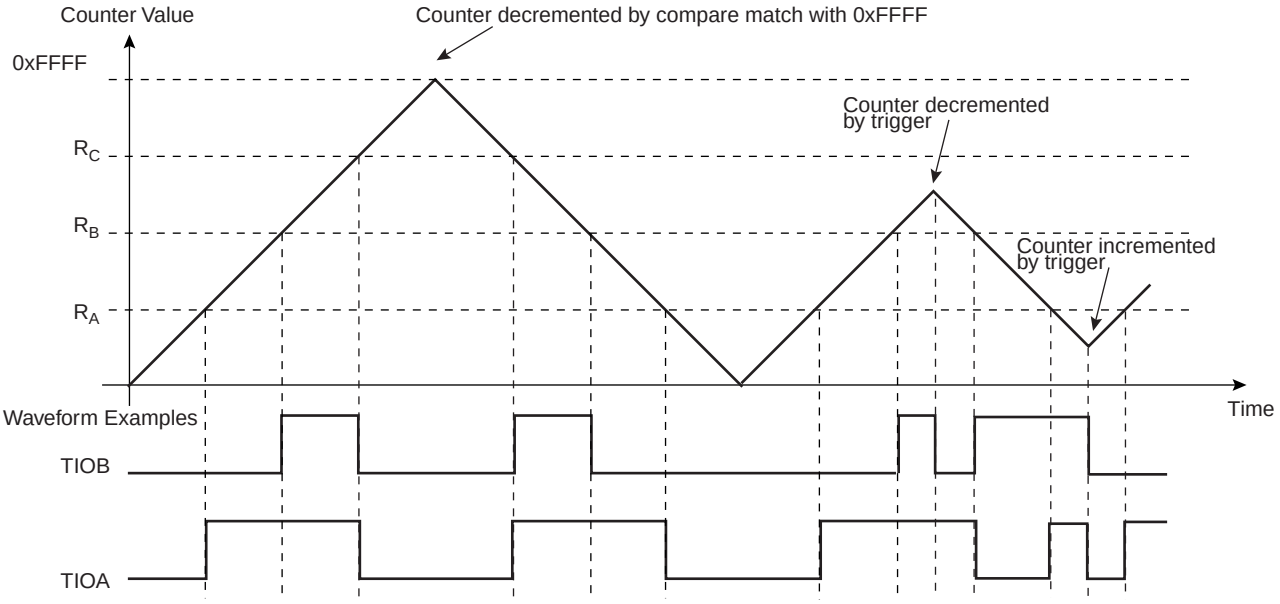


Figure 148. WAVSEL = 01 With Trigger



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 149.

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 150.

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

Figure 149. WAVSEL = 11 Without Trigger

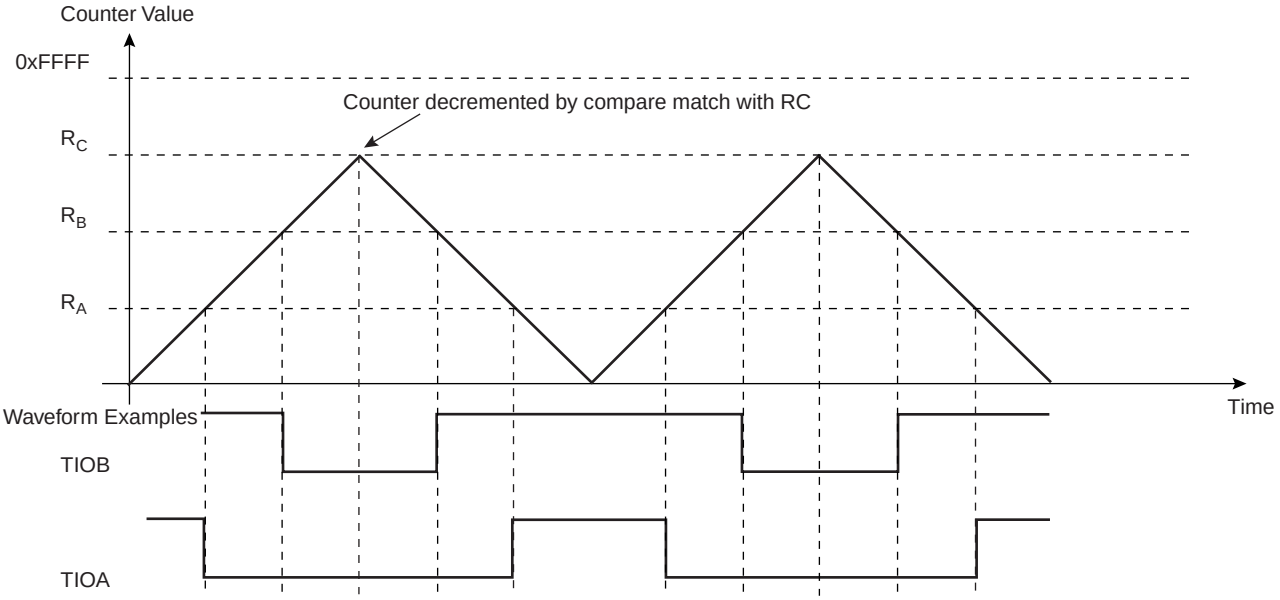
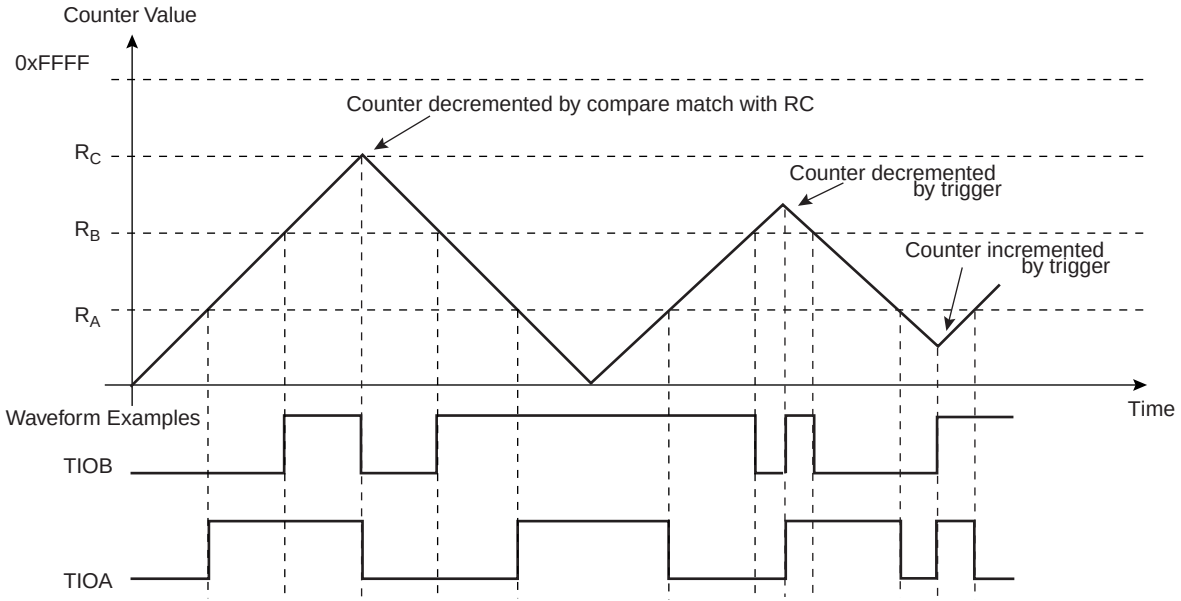


Figure 150. WAVSEL = 11 With Trigger



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 ENETRG 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

Table 61. Timer Counter Global Memory Map

Offset	Channel/Register	Name	Access	Reset Value
0x00	TC Channel 0		See Table 62 See Table 62 See Table 62	
0x40	TC Channel 1			
0x80	TC Channel 2			
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 62. The offset of each of the channel registers in Table 62 is in relation to the offset of the corresponding channel as mentioned in Table 62.

Table 62. Timer Counter Channel Memory Map

Offset	Register	Name	Access	Reset Value
0x00	Channel Control Register	TC_CCR	Write-only	–
0x04	Channel Mode Register	TC_CMR	Read/Write	0
0x08	Reserved			–
0x0C	Reserved			–
0x10	Counter Value	TC_CV	Read-only	0
0x14	Register A	TC_RA	Read/Write(1)	0
0x18	Register B	TC_RB	Read/Write(1)	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

Notes: 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
1	1	TIOA2

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

TC1XC1S		Signal Connected to XC1
0	0	TCLK1
0	1	none
1	0	TIOA0
1	1	TIOA2

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

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

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_CMR

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_CMR

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		ENETRG	EEVT		EEVTEDG	
7	6	5	4	3	2	1	0
CPCDIS	CPCSTOP	BURST		CLKI	TCCLKS		

- **TCCLKS: Clock Selection**

TCCLKS			Clock Selected
0	0	0	TIMER_CLOCK1
0	0	1	TIMER_CLOCK2
0	1	0	TIMER_CLOCK3
0	1	1	TIMER_CLOCK4
1	0	0	TIMER_CLOCK5
1	0	1	XC0
1	1	0	XC1
1	1	1	XC2

- **CLKI: Clock Invert**

0 = Counter is incremented on rising edge of the clock.

1 = Counter is incremented on falling edge of the clock.

- **BURST: Burst Signal Selection**

BURST		
0	0	The clock is not gated by an external signal.
0	1	XC0 is ANDed with the selected clock.
1	0	XC1 is ANDed with the selected clock.
1	1	XC2 is ANDed with the selected clock.

- **CPCSTOP: Counter Clock Stopped with RC Compare**

0 = Counter clock is not stopped when counter reaches RC.

1 = Counter clock is stopped when counter reaches RC.

- **CPCDIS: Counter Clock Disable with RC Compare**

0 = Counter clock is not disabled when counter reaches RC.

1 = Counter clock is disabled when counter reaches RC.

- **EEVTEDG: External Event Edge Selection**

EEVTEDG		Edge
0	0	none
0	1	rising edge
1	0	falling edge
1	1	each edge

- **EEVT: External Event Selection**

EEVT		Signal selected as external event	TIOB Direction
0	0	TIOB	input(1)
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*.

- **ENETRG: External Event Trigger Enable**

0 = The external event has no effect on the counter and its clock. In this case, the selected external event only controls the TIOA output.

1 = The external event resets the counter and starts the counter clock.

- **WAVSEL: Waveform Selection**

WAVSEL		Effect
0	0	UP mode without automatic trigger on RC Compare
1	0	UP mode with automatic trigger on RC Compare
0	1	UPDOWN mode without automatic trigger on RC Compare
1	1	UPDOWN mode with automatic trigger on RC Compare

- **WAVE = 1**

0 = Waveform Mode is disabled (Capture Mode is enabled).

1 = Waveform Mode is enabled.

- **ACPA: RA Compare Effect on TIOA**

ACPA		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **ACPC: RC Compare Effect on TIOA**

ACPC		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

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

AEEVT		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **ASWTRG: Software Trigger Effect on TIOA**

ASWTRG		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **BCPB: RB Compare Effect on TIOB**

BCPB		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **BCPC: RC Compare Effect on TIOB**

BCPC		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **BEEVT: External Event Effect on TIOB**

BEEVT		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

- **BSWTRG: Software Trigger Effect on TIOB**

BSWTRG		Effect
0	0	none
0	1	set
1	0	clear
1	1	toggle

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.

MultiMedia Card Interface (MCI)

Overview

The MultiMedia Card Interface (MCI) supports the MultiMediaCard (MMC) Specification V2.2 and the SD Memory Card Specification V1.0.

The MCI includes a command register, response registers, data registers, timeout counters and error detection logic that automatically handle the transmission of commands and, when required, the reception of the associated responses and data with limited processor overhead.

The MCI supports stream, block and multi-block data read and write, and is compatible with the Peripheral Data Controller channels, minimizing processor intervention for large buffer transfers.

The MCI operates at a rate of up to Master Clock divided by 2 and supports interfacing of up to 16 slots (depending on the product). Each slot may be used to interface with a MultiMediaCard bus (up to 30 Cards) or with an SD Memory Card. Only one slot can be selected at a time (slots are multiplexed). A bit in the Command Register performs this selection.

The SD Memory Card communication is based on a 9-pin interface (clock, command, four data and three power lines) and the MultiMediaCard on a 7-pin interface (clock, command, one data and three power lines).

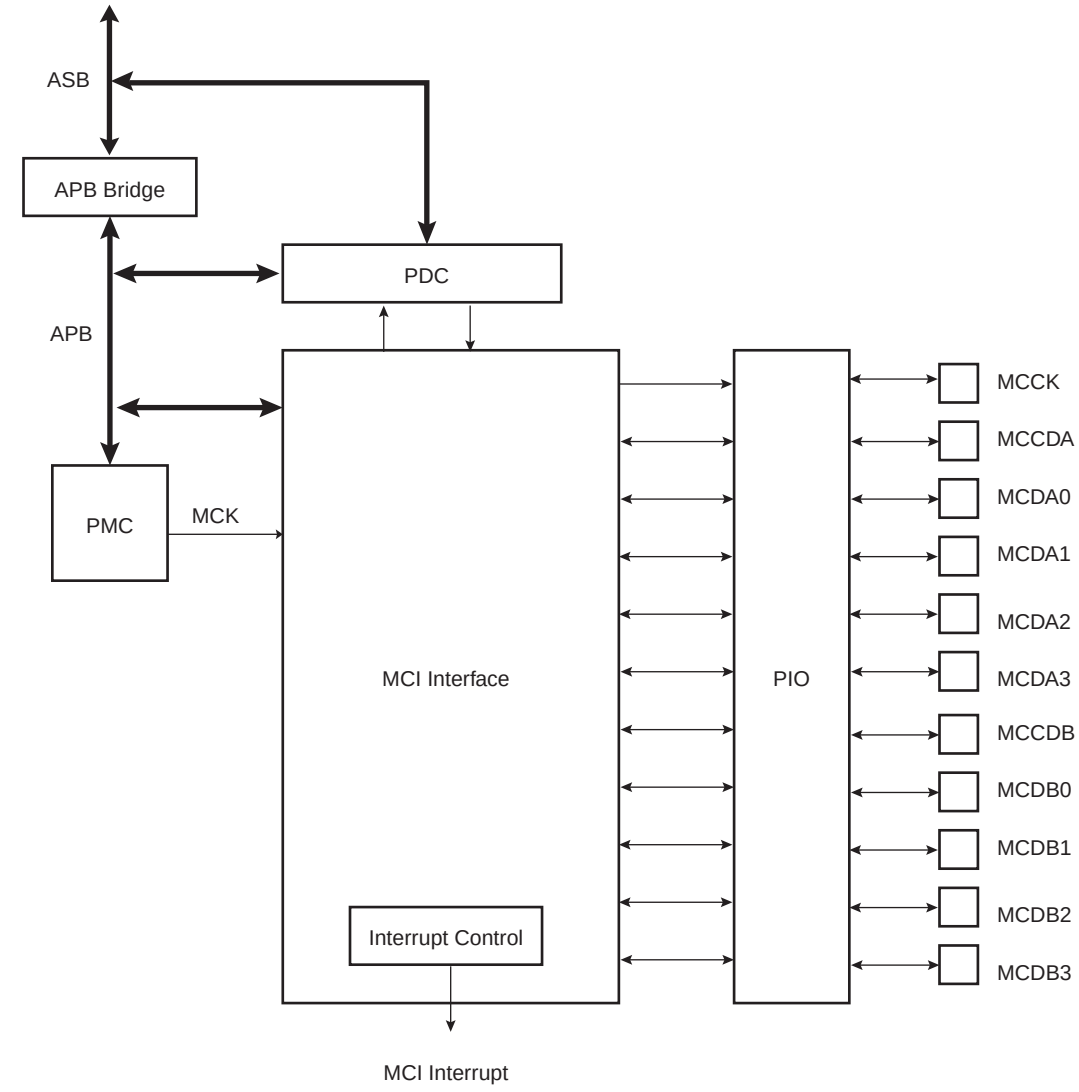
The SD Memory Card interface also supports MultiMedia Card operations. The main differences between SD and MultiMedia Cards are the initialization process and the bus topology.

The main features of the MCI are:

- Compatibility with MultiMedia Card Specification Version 2.2
- Compatibility with SD Memory Card Specification Version 1.0
- Cards clock rate up to Master Clock divided by 2
- Embedded power management to slow down clock rate when not used
- Supports up to sixteen multiplexed slots (product-dependent)
  - One slot for one MultiMediaCard bus (up to 30 cards) or one SD Memory Card
- Support for stream, block and multi-block data read and write
- Supports connection to Peripheral Data Controller
  - Minimizes processor intervention for large buffer transfers

Block Diagram

Figure 151. Block Diagram



Application Block Diagram

Figure 152. Application Block Diagram

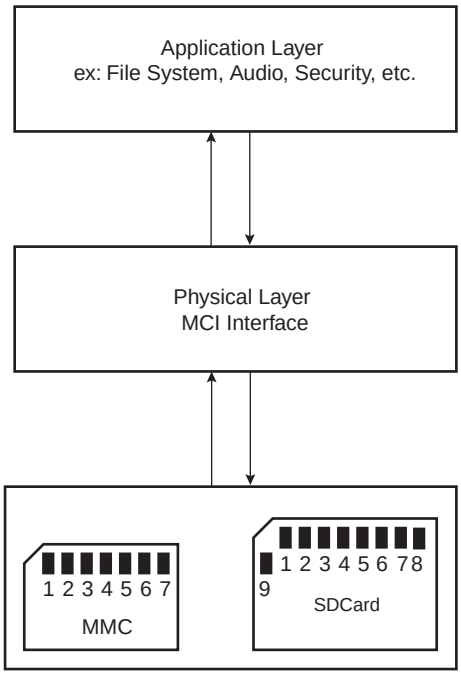


Table 63. I/O Lines Description

Pin Name	Pin Description	Type[1]	Comments
MCCDA/MCCDB	Command/response	I/O/PP/OD	CMD of an MMC or SD Card
MCCK	Clock	I	CLK of an MMC or SD Card
MCDA0 - MCDA3	Data 0..3 of Slot A	I/O/PP	DAT0 of an MMC DAT[0..3] of an SD Card
MCDB0 - MCDB3	Data 0..3 of Slot B	I/O/PP	DAT0 of an MMC DAT[0..3] of an SD Card

Note: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

Product Dependencies

I/O Lines

The pins used for interfacing the MultiMedia Cards or SD Cards may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the peripheral functions to MCI pins.

Power Management

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

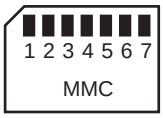
Interrupt

The MCI interface has an interrupt line connected to the Advanced Interrupt Controller (AIC).

Handling the MCI interrupt requires programming the AIC before configuring the MCI.

Bus Topology

Figure 153. MultiMedia Memory Card Bus Topology



The MultiMedia Card communication is based on a 7-pin serial bus interface. It has three communication lines and four supply lines.

Table 64. Bus Topology

Pin Number	Name	Type[1]	Description	MCI Pin Name
1	RSV	NC	Not connected	
2	CMD	I/O/PP/OD	Command/response	MCCDA/MCCDB
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data 0	MCDA0/MCDB0

Note: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.

Figure 154. MMC Bus Connections

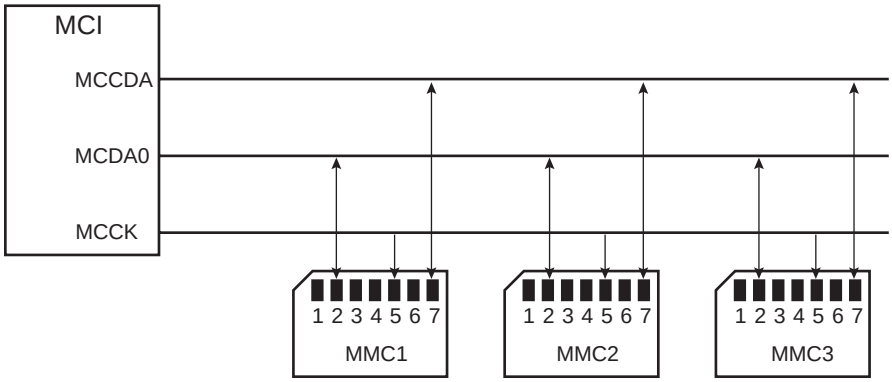
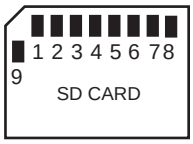


Figure 155. SD Memory Card Bus Topology



The SD Memory Card bus includes the signals listed in Table 65.

Table 65. SD Memory Card Bus Signals

Pin Number	Name	Type[1]	Description	MCI Pin Name
1	CD/DAT[3]	I/O/PP	Card detect/ Data line Bit 3	MCDA3/MCDB3
2	CMD	PP	Command/response	MCCDA/MCCDB
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data line Bit 0	MCDA0/MCDB0
8	DAT[1]	I/O/PP	Data line Bit 1	MCDA1/MCDB1
9	DAT[2]	I/O/PP	Data line Bit 2	MCDA2/MCDB2

Note: 1. I: input, O: output, PP: Push Pull, OD: Open Drain

Figure 156. SD Card Bus Connections

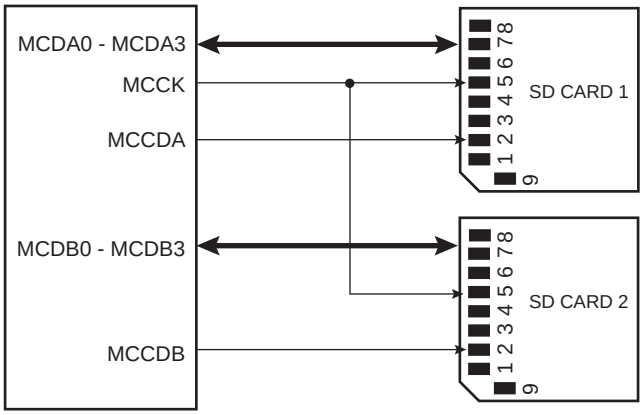
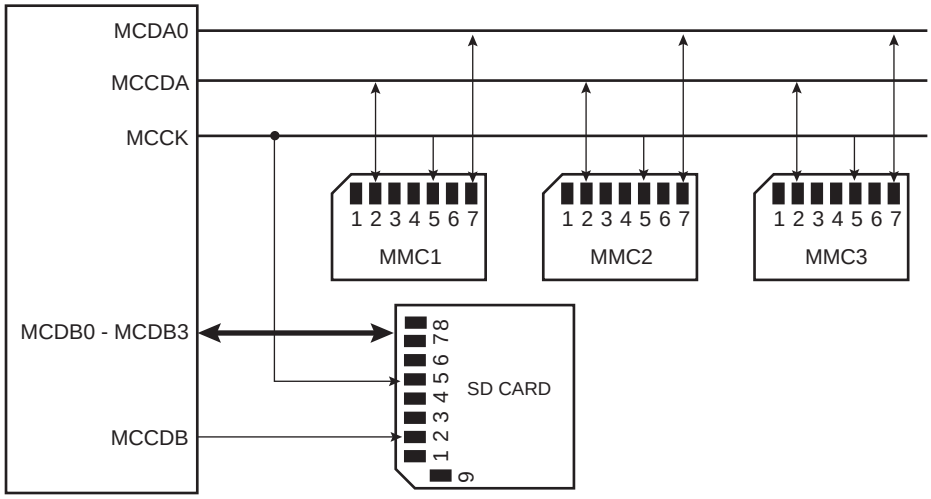


Figure 157. Mixing MultiMedia and SD Memory Cards



When the MCI is configured to operate with SD memory cards, the width of the data bus can be selected in the MCI_SDCR register. Clearing the SDCBUS bit in this register means that the width is one bit and setting it means that the width is four bits. In the case of multimedia cards, only the data line 0 is used. The other data lines can be used as independent PIOs.

MultiMedia Card Operations

After a power-on reset, the cards are initialized by a special message-based MultiMedia Card bus protocol. Each message is represented by one of the following tokens:

- Command: A command is a token that starts an operation. A command is sent from the host either to a single card (addressed command) or to all connected cards (broadcast command). A command is transferred serially on the CMD line.
- Response: A response is a token which is sent from an addressed card or (synchronously) from all connected cards to the host as an answer to a previously received command. A response is transferred serially on the CMD line.
- Data: Data can be transferred from the card to the host or vice versa. Data is transferred via the data line.

Card addressing is implemented using a session address assigned during the initialization phase by the bus controller to all currently connected cards. Their unique CID number identifies individual cards.

The structure of commands, responses and data blocks is described in the MultiMedia-Card System Specification Version 2.2. See also Table 66 on page 377.

MultiMediaCard bus data transfers are composed of these tokens.

There are different types of operations. Addressed operations always contain a command and a response token. In addition, some operations have a data token; the others transfer their information directly within the command or response structure. In this case, no data token is present in an operation. The bits on the DAT and the CMD lines are transferred synchronous to the clock MCCK.

Two types of data transfer commands are defined:

- Sequential commands: These commands initiate a continuous data stream. They are terminated only when a stop command follows on the CMD line. This mode reduces the command overhead to an absolute minimum.
- Block-oriented commands: These commands send a data block succeeded by CRC bits.

Both read and write operations allow either single or multiple block transmission. A multiple block transmission is terminated when a stop command follows on the CMD line similarly to the sequential read.

The MCI provides a set of registers to perform the entire range of MultiMediaCard operations.

Command-response Operation

After reset the MCI is disabled and becomes valid after setting the MCIEN bit in the MCI_CR Control Register. The bit PWSEN allows saving power by dividing the MCI clock by 2 power PWSDIV (MCI_MR) when the bus is inactive.

The command and the response of the card are clocked out with the rising edge of the MCCK.

All the timings for MultiMediaCard are defined in the MultiMediaCard System Specification Version 2.2.

The two bus modes (open drain and push/pull) needed to process all the operations are defined in the MCI command register. The MCI_CMDR allows a command to be carried out.

For example, to perform an ALL_SEND_CID command:

	Host Command					N_{ID} Cycles					CID or OCR			
CMD	S	T	Content	CRC	E	Z	******	Z	S	T	Content	Z	Z	Z

The command ALL_SEND_CID and the fields and values for the MCI_CMDR Control Register are described in Table 66 and Table 67.

Table 66. ALL_SEND_CID command description

CMD Index	Type	Argument	Resp	Abbreviation	Command Description
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the CMD line

Table 67. Fields and Values for MCI_CMDR Command Register

Field	Value
CMDNB (command number)	2 (CMD2)
RSPTYP (response type)	2 (R2: 136 bits response)
SPCMD (special command)	0 (not a special command)
OPCMD (open drain command)	1
MAXLAT (max latency for command to response)	0 (NID cycles ==> 5 cycles)
TRCMD (transfer command)	0 (No transfer)
TRDIR (transfer direction)	X (available only in transfer command)
TRTYP (transfer type)	X (available only in transfer command)

The MCI_ARGR contains the argument field of the command.

To send a command, the user must perform the following steps:

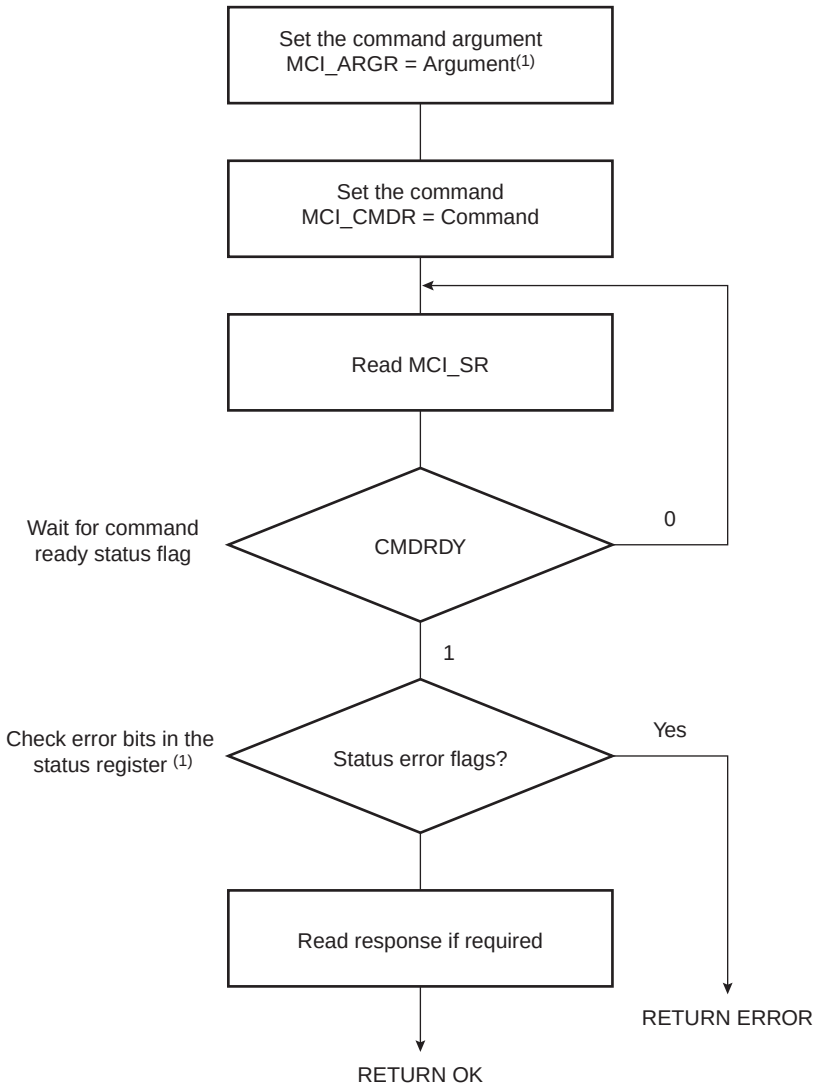
- Fill the argument register (MCI_ARGR) with the command argument.
- Set the command register (MCI_CMDR) (see Table 67).

The command is sent immediately after writing the command register. The status bit CMDRDY in the status register (MCI_SR) is asserted until the command is completed. If the

command requires a response, it can be read in the MCI response register (MCI_RSPR). The response size can be 48 bits up to 136 bits according to the command. The MCI embeds an error detection to prevent any corrupted data during the transfer.

The following flowchart shows how to send a command to the card and read the response if needed. In this example, the status register bits are polled but setting the appropriate bits in the interrupt enable register (MCI_IER) allows using an interrupt method.

Figure 158. Command/Response Functional Flow Diagram



Note: 1. If the command is SEND_OP_COND, the CRC error flag is always present (refer to R3 response in the MultiMediaCard specification).

Data Transfer Operation

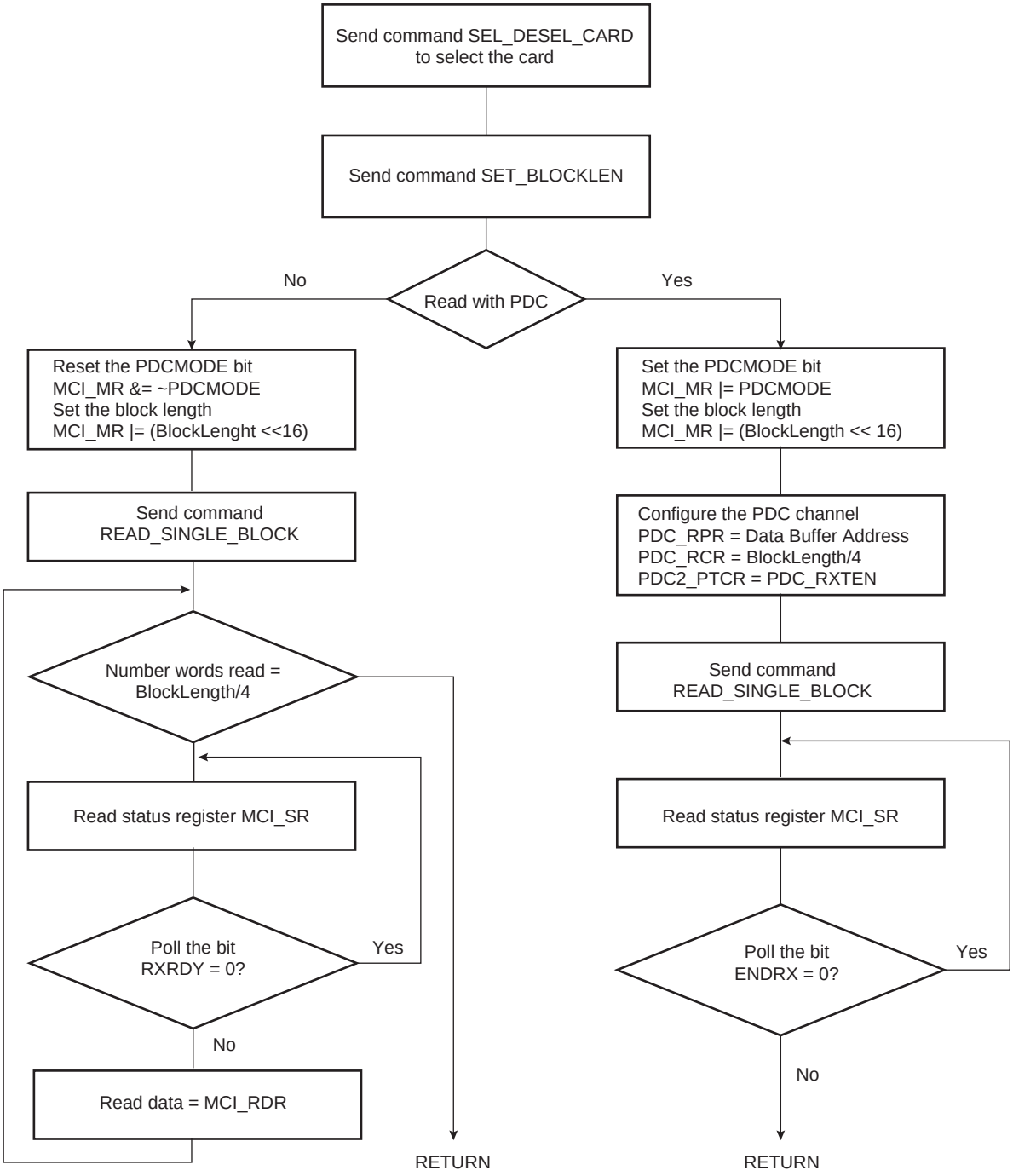
The MultiMedia Card allows several read/write operations (single block, multiple blocks, stream, etc.).

These operations can be done using the Peripheral Data Controller (PDC) features. If the PDCMODE bit is set in MCI_MR, then all reads and writes use the PDC facilities. In all cases, the block length must be defined in the mode register.

Read Operation

The following flowchart shows how to read a single block with or without use of PDC facilities. In this example, a polling method is used to wait for the end of read. Similarly, the user can configure the interrupt enable register (MCI_IER) to trigger an interrupt at the end of read. These two methods can be applied for all MultiMediaCard read functions.

Figure 159. Read Functional Flow Diagram



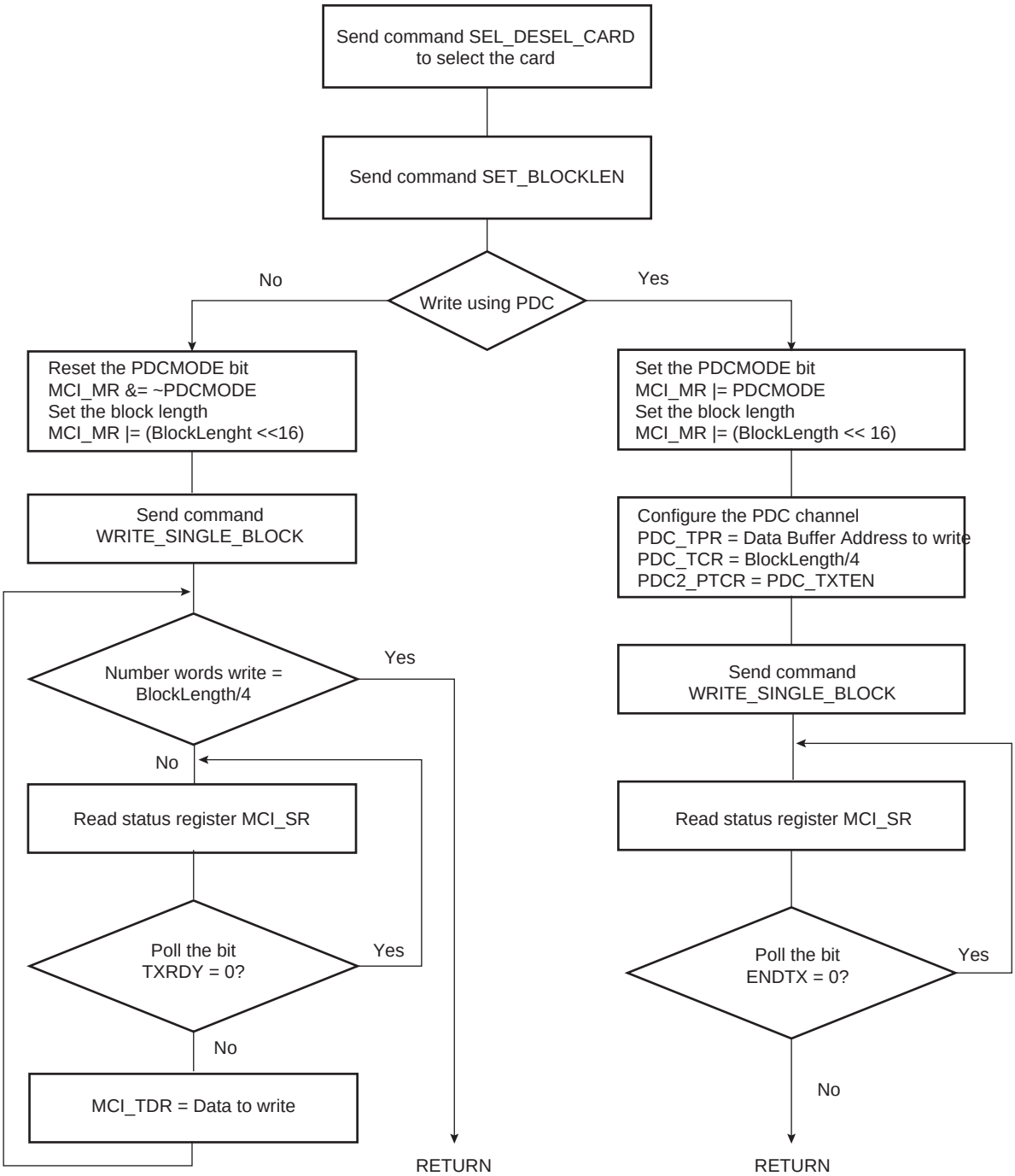
Write Operation

In write operation the MCI Mode Register (MCI_MR) is used to define the padding value when writing non-multiple block size. If the bit PDCPADV is 0, then 0x00 value is used when padding data, otherwise 0xFF is used. If set, the bit PDCMODE enables PDC transfer.

The following flowchart shows how to write a single block with or without use of PDC facilities. Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (MCI_IMR).

This flowchart can be adapted to perform all the MultiMedia Card write functions.

Figure 160. Write Functional Flow Diagram



SD Card Operations

The MultiMedia Card Interface allows processing of SD Memory Card (Secure Digital Memory Card) commands. The SD Memory Card will include a copyright protection mechanism that complies with the security requirements of the SDMI standard, is faster and applicable to higher memory capacity.

The physical form factor, pin assignment and data transfer protocol are forward-com-patible with the MultiMedia Card with some additions.

The SD Memory Card communication is based on a 9-pin interface (Clock, Command, 4 x Data and 3 x Power lines). The communication protocol is defined as a part of this specification. The main difference between the SD Memory Card and the MultiMedia Card is the initialization process.

The SD Card Control Register (MCI_SDCR) allows selection of the card slot and the data bus width.

The SD Card bus allows dynamic configuration of the number of data lines. After power up, by default, the SD Memory Card will use only DAT0 for data transfer. After initialization, the host can change the bus width (number of active data lines).

MultiMedia Card (MCI) User Interface

Table 68. MCI Register Mapping

Offset	Register	Register Name	Read/Write	Reset
0x00	Control Register	MCI_CR	Write	---
0x04	Mode Register	MCI_MR	Read/write	0x0
0x08	Data Timeout Register	MCI_DTOR	Read/write	0x0
0x0C	SD Card Register	MCI_SDCR	Read/write	0x0
0x10	Argument Register	MCI_ARGR	Read/write	0x0
0x14	Command Register	MCI_CMDR	Write	---
0x18 - 0x1C	Reserved			
0x20	Response Register(1)	MCI_RSPR	Read	0x0
0x24	Response Register(1)	MCI_RSPR	Read	0x0
0x28	Response Register(1)	MCI_RSPR	Read	0x0
0x2C	Response Register(1)	MCI_RSPR	Read	0x0
0x30	Receive Data Register	MCI_RDR	Read	0x0
0x34	Transmit Data Register	MCI_TDR	Write	---
0x38 - 0x3C	Reserved			
0x40	Status Register	MCI_SR	Read	0xC0E5
0x44	Interrupt Enable Register	MCI_IER	Write	---
0x48	Interrupt Disable Register	MCI_IDR	Write	---
0x4C	Interrupt Mask Register	MCI_IMR	Read	0x0
0x50-0xFF	Reserved			
0x100-0x124	Reserved for the PDC			

Note: 1. The response register can be read by N accesses at the same MCI_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

MCI Control Register

Register name: MCI_CR

Access Type: Write-only

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

- **MCIEN: Multi-Media Interface Enable**

0 = No effect.

1 = Enables the Multi-Media Interface if MCDIS is 0.

- **MCIDIS: Multi-Media Interface Disable**

0 = No effect.

1 = Disables the Multi-Media Interface.

- **PWSEN: Power Save Mode Enable**

0 = No effect.

1 = Enables the Power Saving Mode if PWSDIS is 0.

- **PWSDIS: Power Save Mode Disable**

0 = No effect.

1 = Disables the Power Saving Mode.

MCI Mode Register

Name: MCI_MR

Access Type: Read/write

31	30	29	28	27	26	25	24
–	–	BLKLEN					

23	22	21	20	19	18	17	16
BLKLEN						0	0

15	14	13	12	11	10	9	8
PDCMODE	PDCPADV	–	–	–	PWSDIV		

7	6	5	4	3	2	1	0
CLKDIV							

- **CLKDIV: Clock Divider**

Multi-Media Card Interface clock (MCCK) is Master Clock (MCK) divided by (2*(CLKDIV+1)).

- **PWSDIV: Power Saving Divider**

Multimedia Card Interface clock is divided by $2^{(PWSDIV)}$ when entering Power Saving Mode.

- **PDCPADV: PDC Padding Value**

0 = 0x00 value is used when padding data in write transfer (not only PDC transfer).

1 = 0xFF value is used when padding data in write transfer (not only PDC transfer).

- **PDCMODE: PDC-oriented Mode**

0 = Disables PDC transfer

1 = Enables PDC transfer. In this case, UNRE and OVRE (MCI_SR) are deactivated.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

Bits 16 and 17 must be 0.

MCI Data Timeout Register

Name: MCI_DTOR

Access Type: Read/write

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

- **DTOCYC: Data Timeout Cycle Number**
- **DTOMUL: Data Timeout Multiplier**

These fields determine the maximum number of Master Clock cycles that the MCI waits between two data block transfers. It equals (DTOCYC x Multiplier).

Multiplier is defined by DTOMUL as shown in the following table:

DTOMUL			Multiplier
0	0	0	1
0	0	1	16
0	1	0	128
0	1	1	256
1	0	0	1024
1	0	1	4096
1	1	0	65536
1	1	1	1048576

MCI SD Card Register

Name: MCI_SDCR

Access Type: Read/write

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

- **SDCSEL: SD Card Selector**

0 = SD card A selected.

1 = SD card B selected.

- **SDCBUS**

0 = 1-bit data bus

1 = 4-bit data bus

MCI Argument Register

Name: MCI_ARGR

Access Type: Read/write

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

- **ARG: Command Argument**

MCI Command Register

Name: MCI_CMDR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	TRTYPE	TRDIR	TRCMD	
15	14	13	12	11	10	9	8
–	–	–	MAXLAT	OPDCMD	SPCMD		
7	6	5	4	3	2	1	0
RSPTYP		CMDNB					

This register is write-protected while CMDRDY is 0 in MCI_SR and in the case of a no Interrupt command sent (bit SPCMD). This means that the current command execution cannot be interrupted or modified.

- **CMDNB: Command Number**
- **RSPTYP: Response Type**

RSP		Response Type
0	0	No response.
0	1	48-bit response.
1	0	136-bit response.
1	1	Reserved.

- **SPCMD: Special CMD**

SPCMD			CMD
0	0	0	Not a special CMD.
0	0	1	Initialization CMD: 74 clock cycles for initialization sequence.
0	1	0	Synchronized CMD: Wait for the end of the current data block transfer before sending the pending command.
0	1	1	Reserved.
1	0	0	Interrupt command: Corresponds to the Interrupt Mode (CMD40).
1	0	1	Interrupt response: Corresponds to the Interrupt Mode (CMD40).

- **OPDCMD: Open Drain Command**

0 = Push pull command

1 = Open drain command

- **MAXLAT: Max Latency for Command to Response**

0 =5-cycle max latency

1 = 64-cycle max latency

- **TRCMD: Transfer Command**

TRCMD		Transfer Type
0	0	No transfer.
0	1	Start Transfer.
1	0	Stop Transfer.
1	1	Reserved.

- **TRDIR: Transfer Direction**

0 = Write

1 = Read

- **TRTYP: Transfer Type**

TRTYP		Transfer Type
0	0	Block.
0	1	Multiple Block.
1	0	Stream.
1	1	Reserved.

MCI SD Response Register

Name: MCI_RSPR

Access Type: Read-only

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

- **RSP: Response**

MCI SD Receive Data Register

Name: MCI_RDR

Access Type: Read-only

31	30	29	28	27	26	25	24
DATA							

23	22	21	20	19	18	17	16
DATA							

15	14	13	12	11	10	9	8
DATA							

7	6	5	4	3	2	1	0
DATA							

- **DATA: Data to Read**

MCI SD Transmit Data Register

Name: MCI_TDR

Access Type: Write-only

31	30	29	28	27	26	25	24
DATA							

23	22	21	20	19	18	17	16
DATA							

15	14	13	12	11	10	9	8
DATA							

7	6	5	4	3	2	1	0
DATA							

- **DATA: Data to Write**

MCI Status Register

Name: MCI_SR

Access Type: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	TCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready**

0 = A command is in progress.

1 = The last command has been sent. Cleared when writing in the MCI_CMDR.

- **RXRDY: Receiver Ready**

0 = Data has not yet been received since the last read of MCI_RDR.

1 = Data has been received since the last read of MCI_RDR.

- **TXRDY: Transmit Ready**

0= The last data written in MCI_TDR has not yet been transferred in the Shift Register.

1= The last data written in MCI_TDR has been transferred in the Shift Register.

- **BLKE: Data Block Ended**

0 = A data block transfer is not yet finished.

1 = A data block transfer has ended. Set at the end of the last block in PDCMODE, otherwise at the end of the first block. Cleared when reading the MCI_SR.

- **DTIP: Data Transfer in Progress**

0 = No data transfer in progress.

1 = The current data transfer is still in progress, including CRC16 calculation. Cleared at the end of the CRC16 calculation.

- **NOTBUSY: Data Not Busy**

0 = The card is not ready for new data transfer.

1 = The card is ready for new data transfer (Data line DAT0 high corresponding to a free data receive buffer in the card).

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in MCI_RCR or MCI_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in MCI_RCR or MCI_RNCR.

- **ENDTX: End of TX Buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in MCI_TCR or MCI_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in MCI_TCR or MCI_TNCR.

- **RXBUFF: RX Buffer Full**

0 = MCI_RCR or MCI_RNCR has a value other than 0.

1 = Both MCI_RCR and MCI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = MCI_TCR or MCI_TNCR has a value other than 0.

1 = Both MCI_TCR and MCI_TNCR have a value of 0.

- **RINDE: Response Index Error**

0 = No error.

1 = A mismatch is detected between the command index sent and the response index received. Cleared when writing in the MCI_CMDR.

- **RDIRE: Response Direction Error**

0 = No error.

1 = The direction bit from card to host in the response has not been detected.

- **RCRCE: Response CRC Error**

0 = No error.

1 = A CRC7 error has been detected in the response. Cleared when writing in the MCI_CMDR.

- **RENDE: Response End Bit Error**

0 = No error.

1 = The end bit of the response has not been detected. Cleared when writing in the MCI_CMDR.

- **RTOE: Response Time-out Error**

0 = No error.

1 = The response time-out set by MAXLAT in the MCI_CMDR has been exceeded. Cleared when writing in the MCI_CMDR.

- **DCRCE: Data CRC Error**

0 = No error.

1 = A CRC16 error has been detected in the last data block. Cleared when sending a new data transfer command.

- **DTOE: Data Time-out Error**

0 = No error.

1 = The data time-out set by DTOCYC and DTOMUL in MCI_DTOR has been exceeded. Cleared when writing in the MCI_CMDR.

- **OVRE: Overrun**

0 = No error.

1 = At least one 8-bit received data has been lost (not read). Cleared when sending a new data transfer command.

- **UNRE: Underrun**

0 = No error.

1 = At least one 8-bit data has been sent without valid information (not written). Cleared when sending a new data transfer command.

MCI Interrupt Enable Register

Name: MCI_IER

Access Type: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	TCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready Interrupt Enable**
- **RXRDY: Receiver Ready Interrupt Enable**
- **TXRDY: Transmit Ready Interrupt Enable**
- **BLKE: Data Block Ended Interrupt Enable**
- **DTIP: Data Transfer in Progress Interrupt Enable**
- **NOTBUSY: Data Not Busy Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **RINDE: Response Index Error Interrupt Enable**
- **RDIRE: Response Direction Error Interrupt Enable**
- **RCRCE: Response CRC Error Interrupt Enable**
- **RENDE: Response End Bit Error Interrupt Enable**
- **RTOE: Response Time-out Error Interrupt Enable**
- **DCRCE: Data CRC Error Interrupt Enable**
- **DTOE: Data Time-out Error Interrupt Enable**
- **OVRE: Overrun Interrupt Enable**
- **UNRE: UnderRun Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

MCI Interrupt Disable Register

Name: MCI_IDR

Access Type: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	TCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready Interrupt Disable**
- **RXRDY: Receiver Ready Interrupt Disable**
- **TXRDY: Transmit Ready Interrupt Disable**
- **BLKE: Data Block Ended Interrupt Disable**
- **DTIP: Data Transfer in Progress Interrupt Disable**
- **NOTBUSY: Data Not Busy Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **RINDE: Response Index Error Interrupt Disable**
- **RDIRE: Response Direction Error Interrupt Disable**
- **RCRCE: Response CRC Error Interrupt Disable**
- **RENDE: Response End Bit Error Interrupt Disable**
- **RTOE: Response Time-out Error Interrupt Disable**
- **DCRCE: Data CRC Error Interrupt Disable**
- **DTOE: Data Time-out Error Interrupt Disable**
- **OVRE: Overrun Interrupt Disable**
- **UNRE: UnderRun Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

MCI Interrupt Mask Register

Name: MCI_IMR

Access Type: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DTOE	TCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	–	–	–	–	–	–
7	6	5	4	3	2	1	0
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready Interrupt Mask**
- **RXRDY: Receiver Ready Interrupt Mask**
- **TXRDY: Transmit Ready Interrupt Mask**
- **BLKE: Data Block Ended Interrupt Mask**
- **DTIP: Data Transfer in Progress Interrupt Mask**
- **NOTBUSY: Data Not Busy Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **RINDE: Response Index Error Interrupt Mask**
- **RDIRE: Response Direction Error Interrupt Mask**
- **RCRCE: Response CRC Error Interrupt Mask**
- **RENDE: Response End Bit Error Interrupt Mask**
- **RTOE: Response Time-out Error Interrupt Mask**
- **DCRCE: Data CRC Error Interrupt Mask**
- **DTOE: Data Time-out Error Interrupt Mask**
- **OVRE: Overrun Interrupt Mask**
- **UNRE: UnderRun Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

USB Device Port (UDP)

Overview

The USB Device Port (UDP) is compliant with the Universal Serial Bus (USB) V2.0 full-speed device specification. It is designed to be associated with Atmel's embedded USB transceiver and interfaced with an ARM7TDMI and ARM9TDMI core.

The number and size of endpoints is product-dependent. Each endpoint is associated with one or two banks of a dual-port RAM used to store the current data payload. If two banks are used, one DPR bank is read or written by the processor, while the other is read or written by the USB device peripheral. This feature is mandatory for isochronous endpoints. Thus the device maintains the maximum bandwidth (1M bytes/s) by working with endpoints with two banks of DPR.

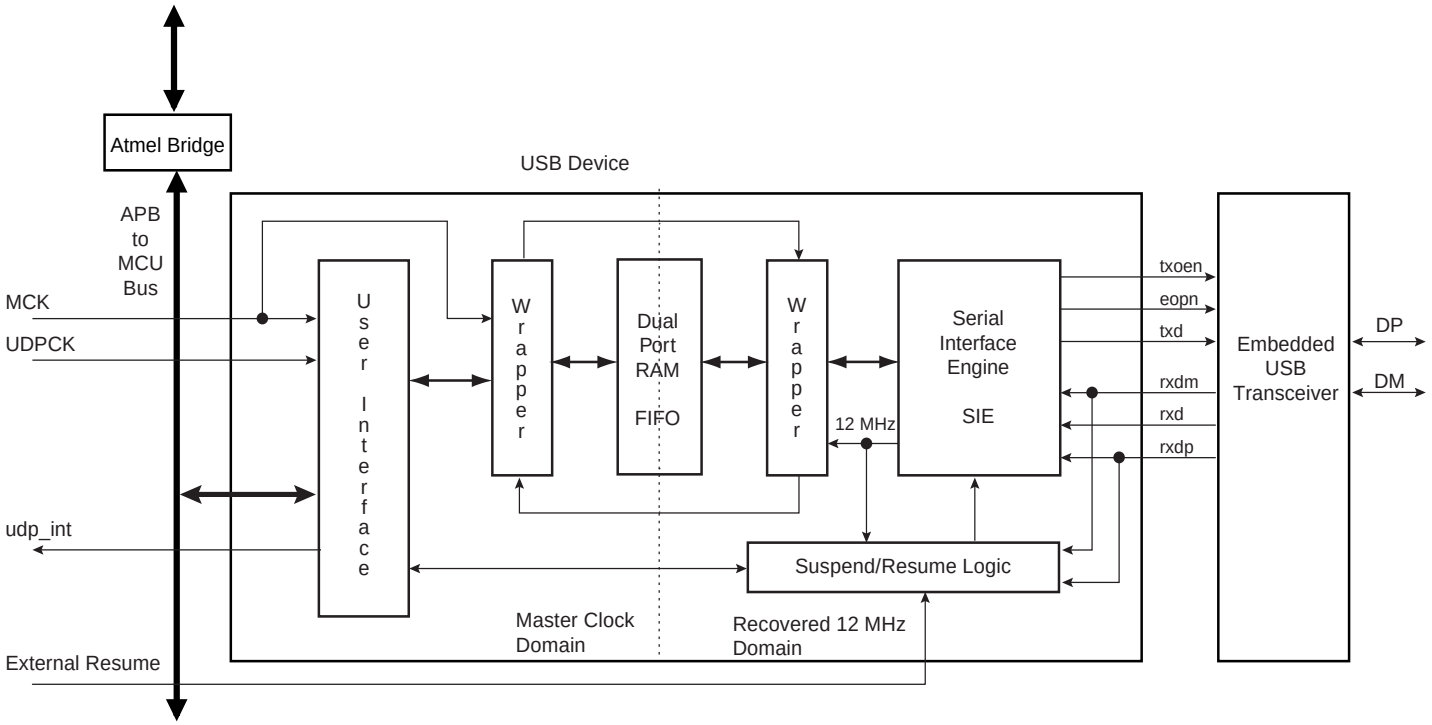
Suspend and resume are automatically detected by the USB device, which notifies the processor by raising an interrupt. Depending on the product, an external signal can be used to send a wake-up to the USB host controller.

The main features of the UDP are:

- USB V2.0 Full-speed Compliant, 12 Mbits per second
- Embedded USB V2.0 Full-speed Transceiver
- Embedded Dual-port RAM for Endpoints
- Suspend/Resume Logic
- Ping-pong Mode (2 Memory Banks) for Isochronous and Bulk Endpoints

Block Diagram

Figure 161. USB Device Port Block Diagram



Access to the UDP is via the APB bus interface. Read and write to the data FIFO are done by reading and writing 8-bit values to APB registers.

The UDP peripheral requires two clocks: one peripheral clock used by the MCK domain and a 48 MHz clock used by the 12 MHz domain.

A USB 2.0 full-speed pad is embedded and controlled by the SIE.

The signal external_resume is optional. It allows the UDP peripheral to wake-up once in system mode. The host will then be notified that the device asks for a resume. This optional feature must be also negotiated with the host during the enumeration.

Product Dependencies

Note: For further details on the USB Device hardware implementation, see the specific Product Properties document.

The USB physical transceiver is integrated into the product. The bi-directional differential signals DP and DM are available from the product boundary.

Two I/O lines may be used by the application:

- One to check that VBUS is still available from the host. Self-powered devices may use this entry to be notified that the host has been powered off. In this case, the board pull-up on DP must be disabled in order to prevent feeding current to the host.
- One to control the board pull-up on DP. Thus, when the device is ready to communicate with the host, it activates its DP pull-up through this control line.

I/O Lines

DP and DM are not controlled by any PIO controllers. The embedded USB physical transceiver is controlled by the USB device peripheral.

To reserve an I/O line to check VBUS, the programmer must first program the PIO controller to assign this I/O in input PIO mode.

To reserve an I/O line to control the board pull-up, the programmer must first program the PIO controller to assign this I/O in output PIO mode.

Power Management

The USB device peripheral requires a 48 MHz clock. This clock must be generated by a PLL with an accuracy of ± 0.25%.

Thus, the USB device receives two clocks from the Power Management Controller (PMC): the master clock, MCK, used to drive the peripheral user interface and the UDPCK used to interface with the bus USB signals (recovered 12 MHz domain).

Interrupt

The USB device interface has an interrupt line connected to the Advanced Interrupt Controller (AIC).

Handling the USB device interrupt requires programming the AIC before configuring the UDP.

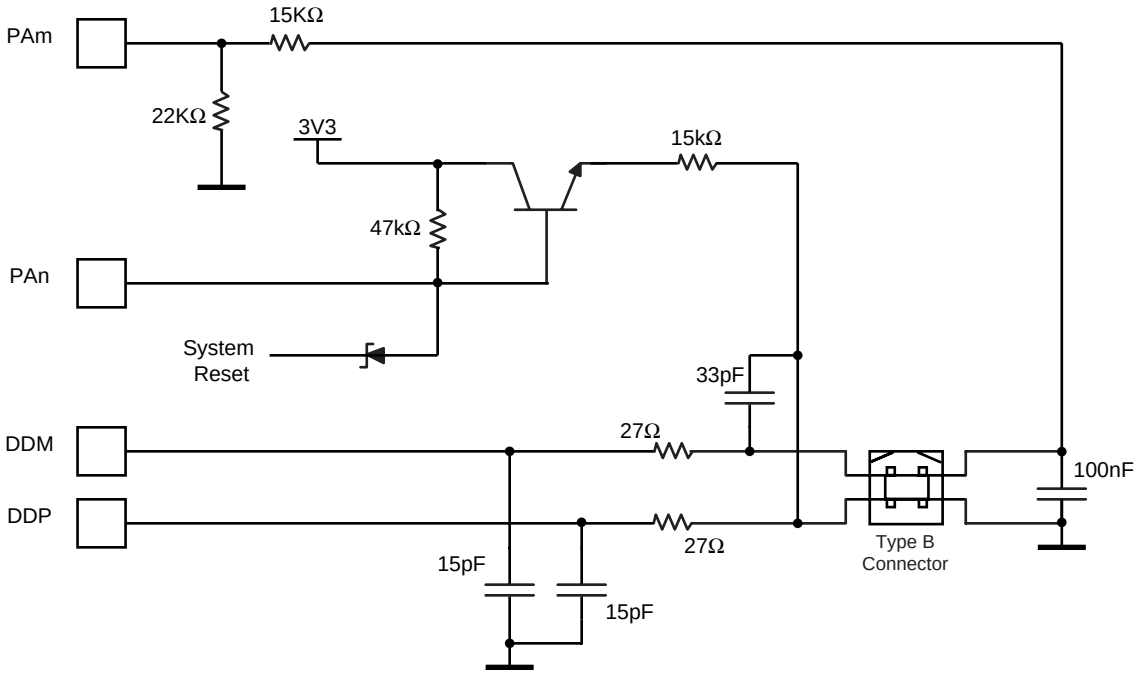
Typical Connection

USB_CNX is an input signal used to check if the host is connected

USB_DP_PUP is an output signal used to enable pull-up on DP.

Figure 162 shows automatic activation of pull-up after reset.

Figure 162. Board Schematic to Interface USB Device Peripheral

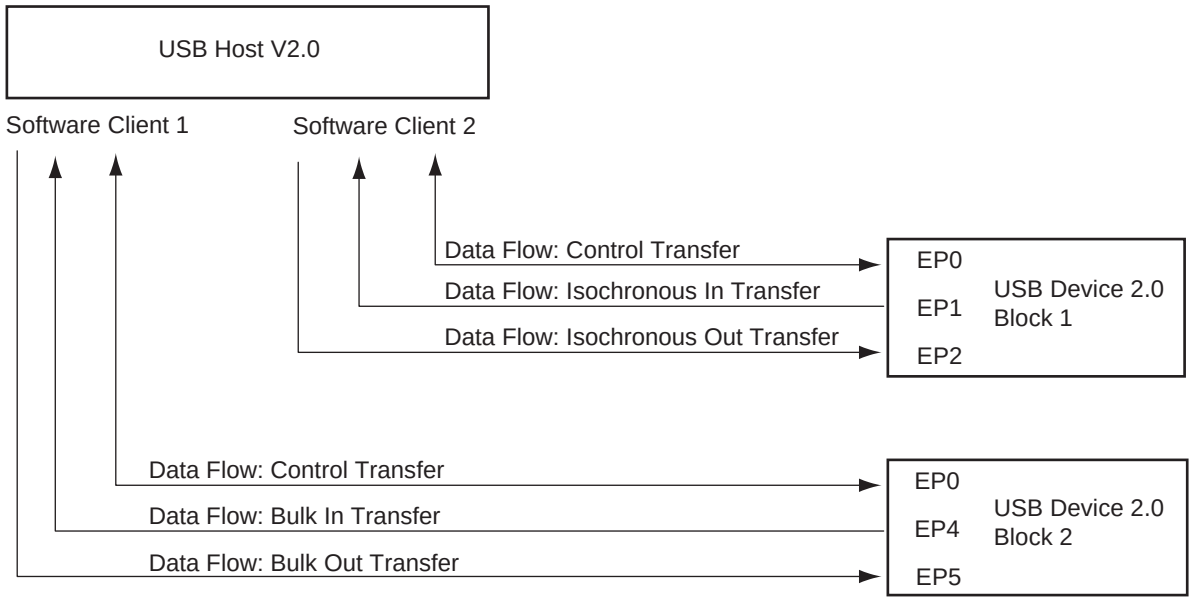


Functional Description

USB V2.0 Full-speed Introduction

The USB V2.0 full-speed provides communication services between host and attached USB devices. Each device is offered with a collection of communication flows (pipes) associated with each endpoint. Software on the host communicates with an USB device through a set of communication flows.

Figure 163. Example of USB V2.0 Full-speed Communication Control



USB V2.0 Full-speed Transfer Types

A communication flow is carried over one of four transfer types defined by the USB device.

Table 69. USB Communication Flow

Transfer	Direction	Bandwidth	Endpoint Size	Error Detection	Retrying
Control	Bi-directional	Not guaranteed	8, 16, 32, 64	Yes	Automatic
Isochronous	Uni-directional	Guaranteed	1 - 1023	Yes	No
Interrupt	Uni-directional	Not guaranteed	≤ 64	Yes	Yes
Bulk	Uni-directional	Not guaranteed	8, 16, 32, 64	Yes	Yes

USB Bus Transactions

Each transfer results in one or more transactions over the USB bus. There are five kinds of transactions flowing across the bus in packets:

1. Setup Transaction
2. Data IN Transaction
3. Data OUT Transaction
4. Status IN Transaction
5. Status OUT Transaction

USB Transfer Event Definitions

As shown in Table 70, transfers are sequential events carried out on the USB bus.

Table 70. USB Transfer Events

Control Transfers(1) (3)	• Setup transaction > Data IN transactions > Status OUT transaction • Setup transaction > Data OUT transactions > Status IN transaction • Setup transaction > Status IN transaction
Interrupt IN Transfer (device toward host)	• Data IN transaction > Data IN transaction
Interrupt OUT Transfer (host toward device)	• Data OUT transaction > Data OUT transaction
Isochronous IN Transfer(2) (device toward host)	• Data IN transaction > Data IN transaction
Isochronous OUT Transfer(2) (host toward device)	• Data OUT transaction > Data OUT transaction
Bulk IN Transfer (device toward host)	• Data IN transaction > Data IN transaction
Bulk OUT Transfer (host toward device)	• Data OUT transaction > Data OUT transaction

Notes:
1. Control transfer must use endpoints with no ping-pong attributes.
2. Isochronous transfers must use endpoints with ping-pong attributes.
3. Control transfers can be aborted using a stall handshake.

Handling Transactions with USB V2.0 Device Peripheral

Setup Transaction

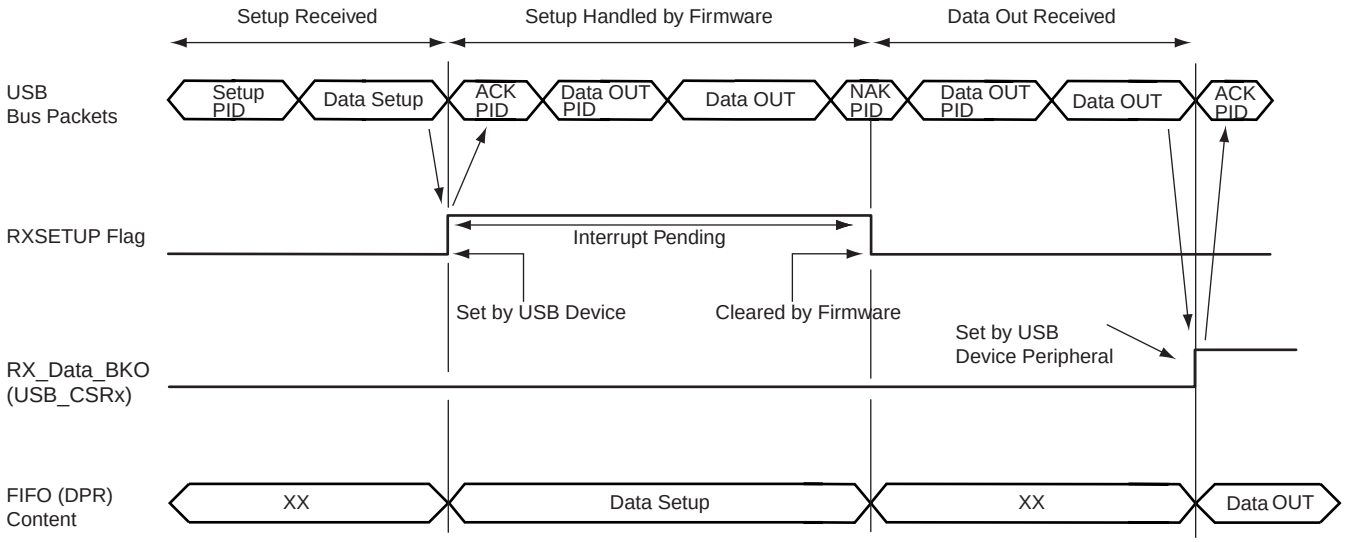
Setup is a special type of host-to-device transaction used during control transfers. Control transfers must be performed using endpoints with no ping-pong attributes. A setup transaction needs to be handled as soon as possible by the firmware. It is used to transmit requests from the host to the device. These requests are then handled by the USB device and may require more arguments. The arguments are sent to the device by a Data OUT transaction which follows the setup transaction. These requests may also return data. The data is carried out to the host by the next Data IN transaction which follows the setup transaction. A status transaction ends the control transfer.

When a setup transfer is received by the USB endpoint:

- The USB device automatically acknowledges the setup packet
- RXSETUP is set in the USB_CSRx register
- An endpoint interrupt is generated while the RXSETUP is not cleared. This interrupt is carried out to the microcontroller if interrupts are enabled for this endpoint.

Thus, firmware must detect the RXSETUP polling the USB_CSRx or catching an interrupt, read the setup packet in the FIFO, then clear the RXSETUP. RXSETUP cannot be cleared before the setup packet has been read in the FIFO. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the setup packet in the FIFO.

Figure 164. Setup Transaction Followed by a Data OUT Transaction



Data IN Transaction

Data IN transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the device to the host. Data IN transactions in isochronous transfer must be done using endpoints with ping-pong attributes.

Using Endpoints Without Ping-pong Attributes

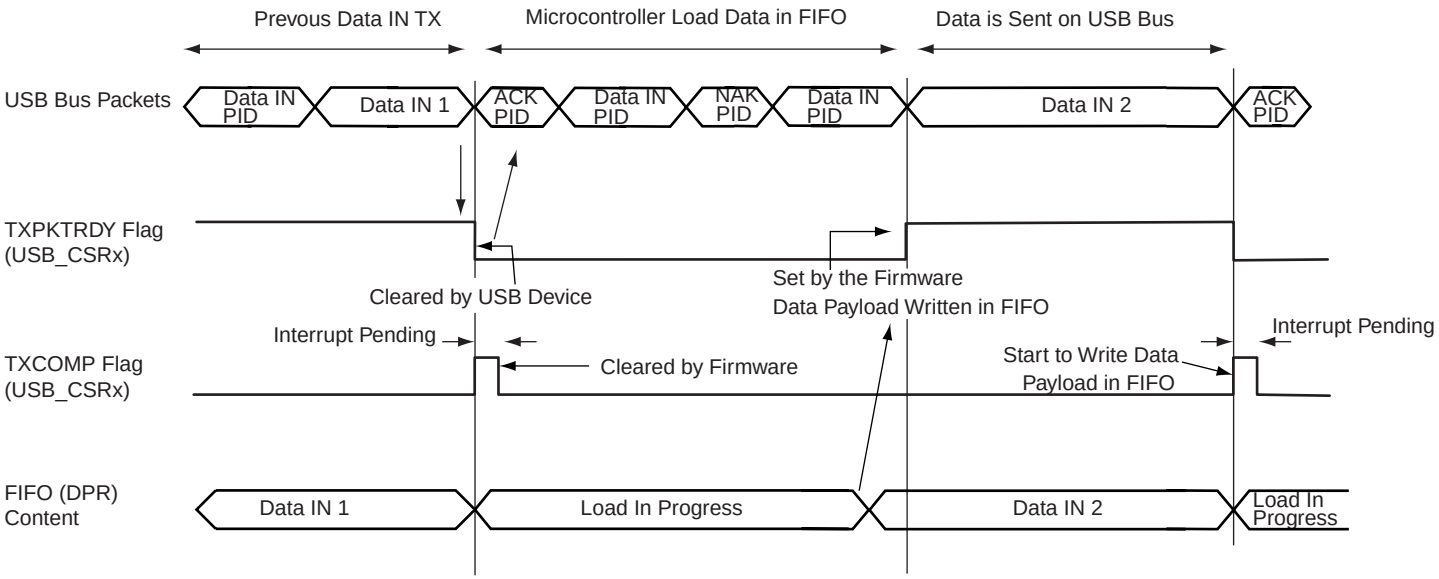
To perform a Data IN transaction, using a non ping-pong endpoint:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY in the endpoint's USB_CSRx register (TXPKTRDY must be cleared).
2. The microcontroller writes data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's USB_FDRx register,
3. The microcontroller notifies the USB peripheral it has finished by setting the TXPK-TRDY in the endpoint's USB_CSRx register,
4. The microcontroller is notified that the endpoint's FIFO has been released by the USB device when TXCOMP in the endpoint's USB_CSRx register has been set. Then an interrupt for the corresponding endpoint is pending while TXCOMP is set.

TXCOMP is set by the USB device when it has received an ACK PID signal for the Data IN packet. An interrupt is pending while TXCOMP is set.

Note: Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the Data IN protocol layer.

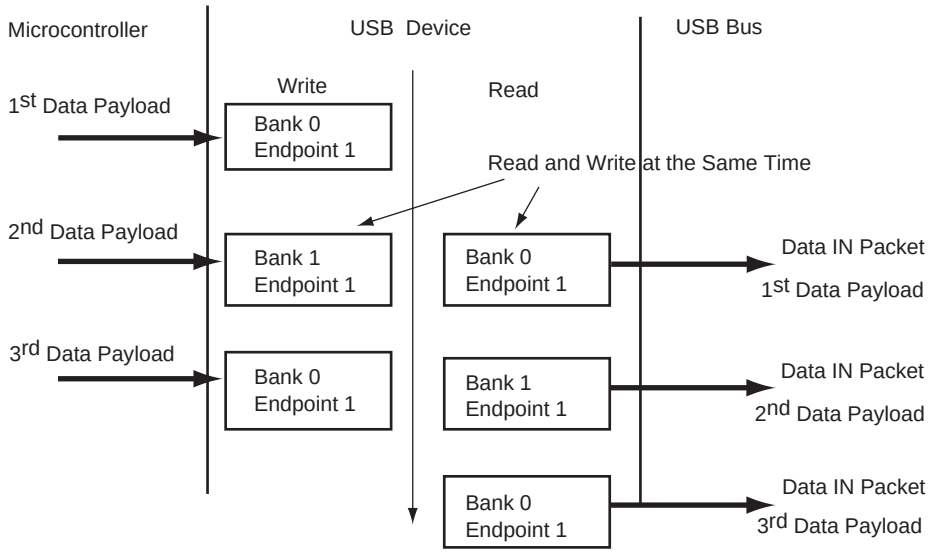
Figure 165. Data IN Transfer for Non Ping-pong Endpoint



Using Endpoints With Ping-pong Attribute

The use of an endpoint with ping-pong attributes is necessary during isochronous transfer. To be able to guarantee a constant bandwidth, the microcontroller must prepare the next data payload to be sent while the current one is being sent by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

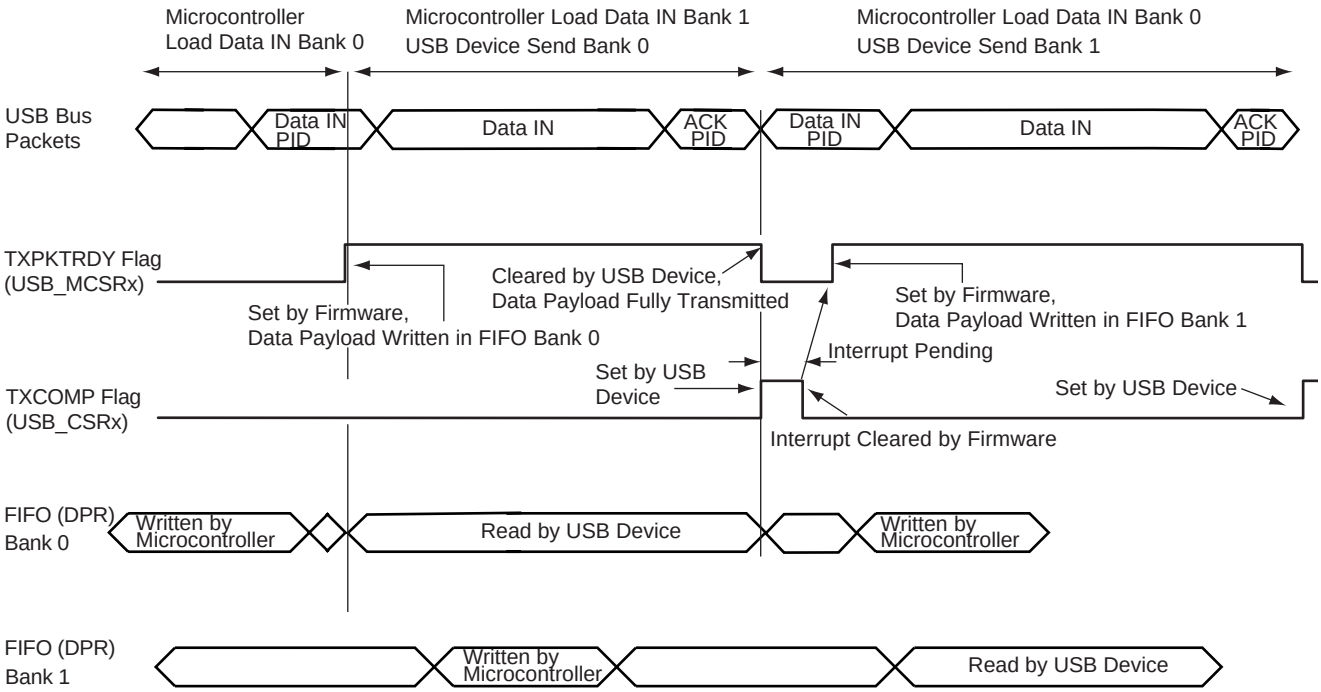
Figure 166. Bank Swapping Data IN Transfer for Ping-pong Endpoints



When using a ping-pong endpoint, the following procedures are required to perform Data IN transactions:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY to be cleared in the endpoint's USB_CSRx register.
2. The microcontroller writes the first data payload to be sent in the FIFO (Bank 0), writing zero or more byte values in the endpoint's USB_FDRx register.
3. The microcontroller notifies the USB peripheral it has finished writing in Bank 0 of the FIFO by setting the TXPKTRDY in the endpoint's USB_CSRx register.
4. Without waiting for TXPKTRDY to be cleared, the microcontroller writes the second data payload to be sent in the FIFO (Bank 1), writing zero or more byte values in the endpoint's USB_FDRx register.
5. The microcontroller is notified that the first Bank has been released by the USB device when TXCOMP in the endpoint's USB_CSRx register is set. An interrupt is pending while TXCOMP is being set.
6. Once the microcontroller has received TXCOMP for the first Bank, it notifies the USB device that it has prepared the second Bank to be sent rising TXPKTRDY in the endpoint's USB_CSRx register.
7. At this step, Bank 0 is available and the microcontroller can prepare a third data payload to be sent.

Figure 167. Data IN Transfer for Ping-pong Endpoint



Warning: There is software critical path due to the fact that once the second bank is filled, the driver has to wait for TX_COMP to set TX_PKTRDY. If the delay between receiving TX_COMP is set and TX_PKTRDY is set is too long, some Data IN packets may be NACKed, reducing the bandwidth.

Data OUT Transaction

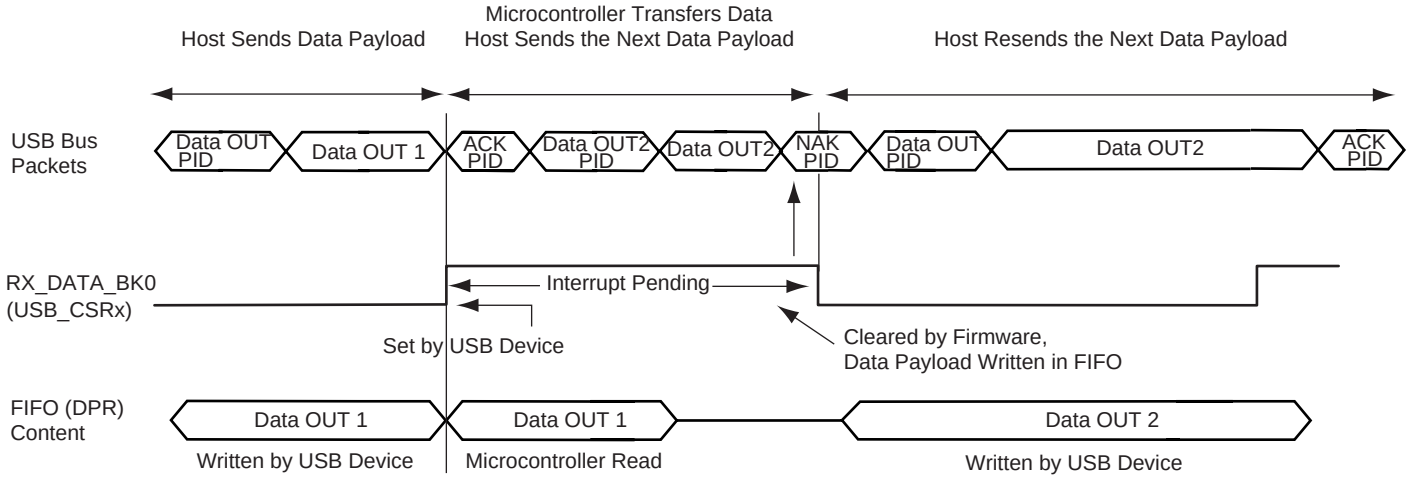
Data OUT transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the host to the device. Data OUT transactions in isochronous transfers must be done using endpoints with ping-pong attributes.

Data OUT Transaction Without Ping-pong Attributes

To perform a Data OUT transaction, using a non ping-pong endpoint:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX_DATA_BK0 in the endpoint's USB_CSRx register. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.
4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's USB_CSRx register.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's USB_FDRx register.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's USB_CSRx register.
7. A new Data OUT packet can be accepted by the USB device.

Figure 168. Data OUT Transfer for Non Ping-pong Endpoints

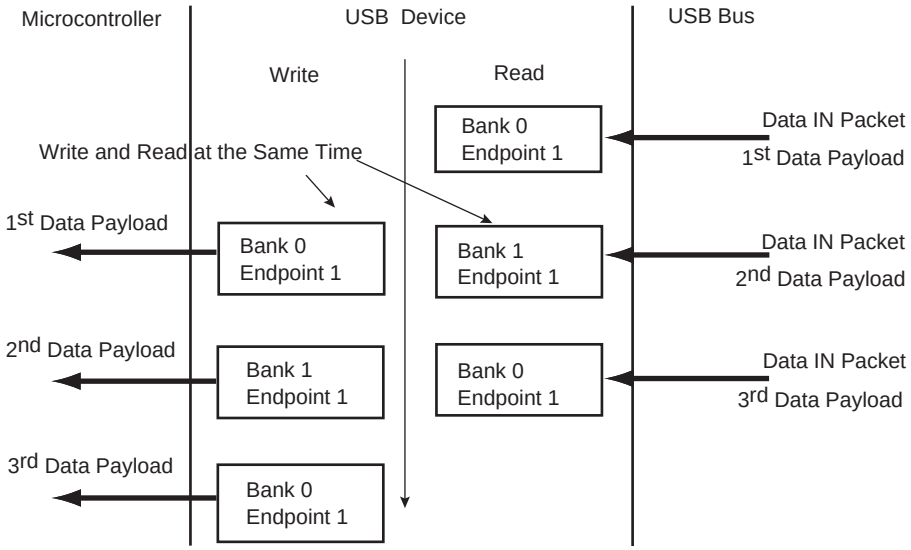


An interrupt is pending while the flag RX_DATA_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory can not be done after RX_DATA_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is necessary. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

Figure 169. Bank Swapping in Data OUT Transfers for Ping-pong Endpoints

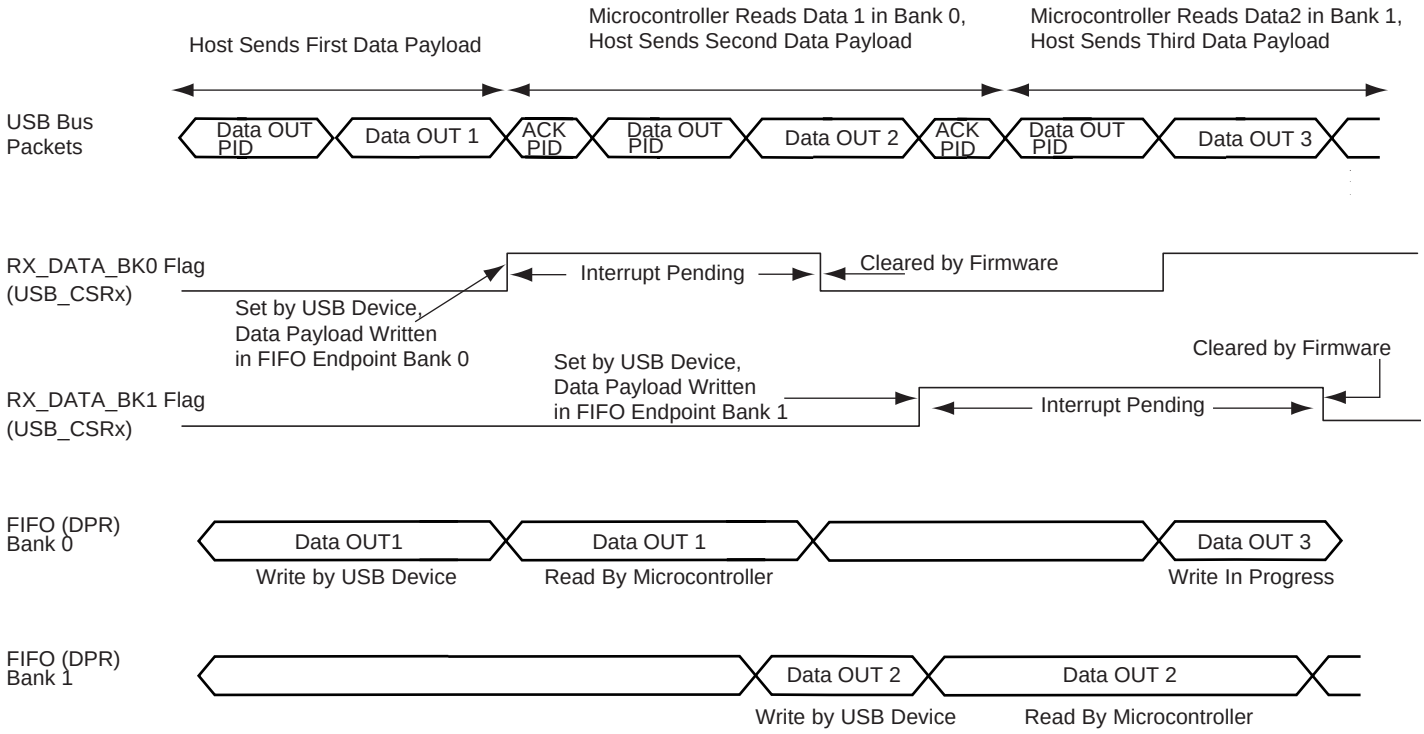


When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling RX_DATA_BK0 in the endpoint's USB_CSRx register. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.
5. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's USB_CSRx register.
6. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is made available by reading the endpoint's USB_FDRx register.
7. The microcontroller notifies the USB peripheral device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's USB_CSRx register.
8. A third Data OUT packet can be accepted by the USB peripheral device and copied in the FIFO Bank 0.
9. If a second Data OUT packet has been received, the microcontroller is notified by the flag RX_DATA_BK1 set in the endpoint's USB_CSRx register. An interrupt is pending for this endpoint while RX_DATA_BK1 is set.

10. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is available by reading the endpoint's USB_FDRx register.
11. The microcontroller notifies the USB device it has finished the transfer by clearing RX_DATA_BK1 in the endpoint's USB_CSRx register.
12. A fourth Data OUT packet can be accepted by the USB device and copied in the FIFO Bank 0.

Figure 170. Data OUT Transfer for Ping-pong Endpoint



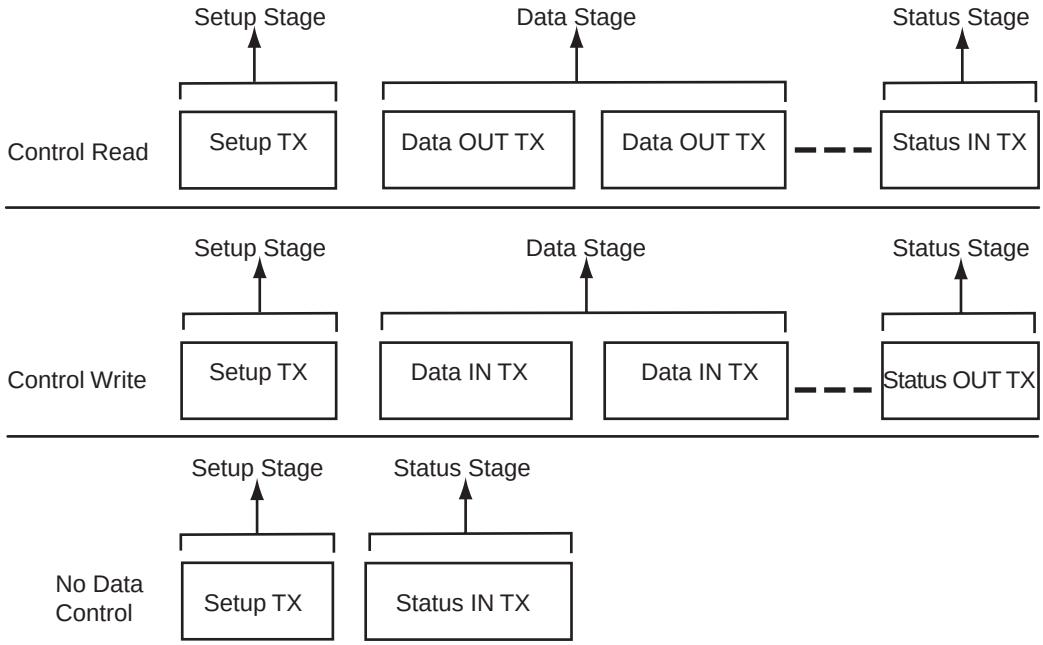
Note: An interrupt is pending while the RX_DATA_BK0 or RX_DATA_BK1 flag is set.

Warning: When RX_DATA_BK0 and RX_DATA_BK1 are both set, there is no way to determine which one to clear first. Thus the software must keep an internal counter to be sure to clear alternatively RX_DATA_BK0 then RX_DATA_BK1. This situation may occur when the software application is busy elsewhere and the two banks are filled by the USB host. Once the application comes back to the USB driver, the two flags are set.

Status Transaction

A status transaction is a special type of host to device transaction used only in a control transfer. The control transfer must be performed using endpoints with no ping-pong attributes. According to the control sequence (read or write), the USB device sends or receives a status transaction.

Figure 171. Control Read and Write Sequences



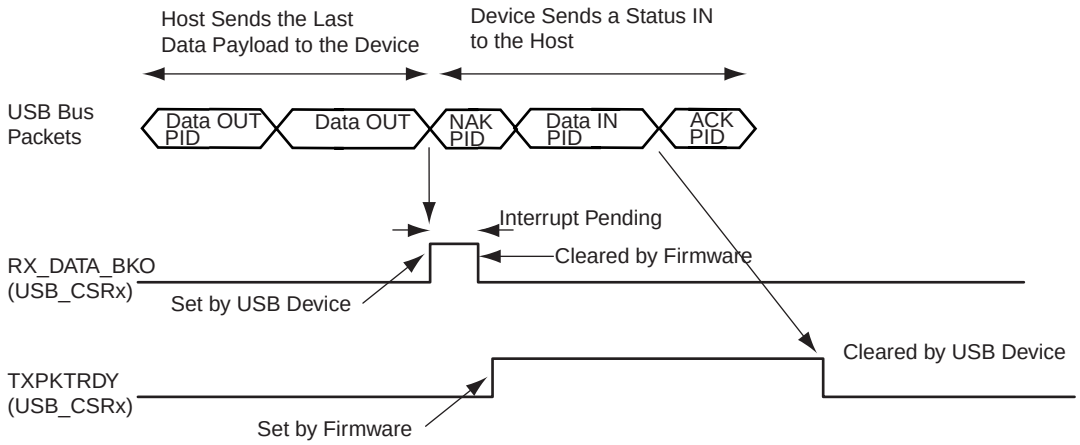
Notes:
1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Please refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0,* to get more information on the protocol layer.
2. During the Status OUT stage, the host emits a zero length packet to the device (Data OUT transaction with no data).

Status IN Transfer

Once a control request has been processed, the device returns a status to the host. This is a zero length Data IN transaction.

1. The microcontroller waits for TXPKTRDY in the USB_CSRx endpoint's register to be cleared. (At this step, TXPKTRDY must be cleared because the previous transaction was a setup transaction or a Data OUT transaction.)
2. Without writing anything to the USB_FDRx endpoint's register, the microcontroller sets TXPKTRDY. The USB device generates a Data IN packet using DATA1 PID.
3. This packet is acknowledged by the host and TXPKTRDY is set in the USB_CSRx endpoint's register.

Figure 172. Data Out Followed by Status IN Transfer.

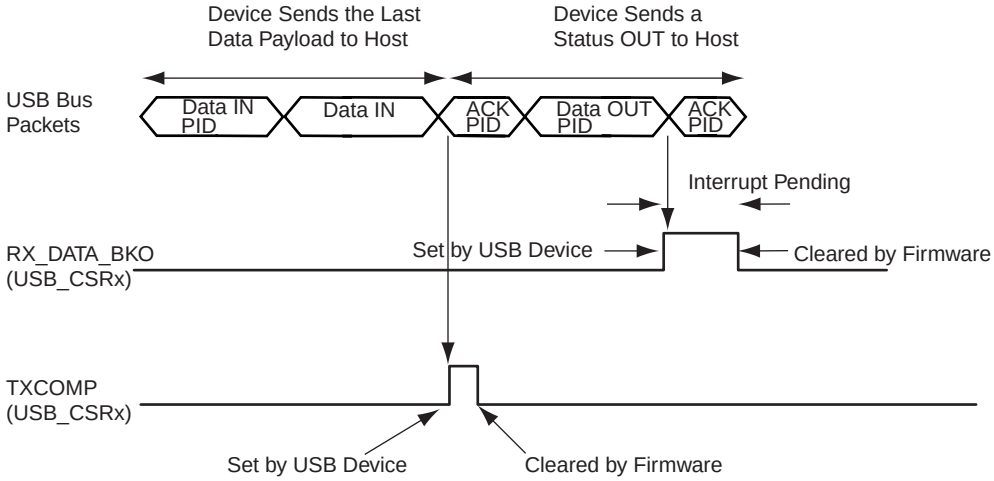


Status OUT Transfer

Once a control request has been processed and the requested data returned, the host acknowledges by sending a zero length packet. This is a zero length Data OUT transaction.

1. The USB device receives a zero length packet. It sets RX_DATA_BK0 flag in the USB_CSRx register and acknowledges the zero length packet.
2. The microcontroller is notified that the USB device has received a zero length packet sent by the host polling RX_DATA_BK0 in the USB_CSRx register. An interrupt is pending while RX_DATA_BK0 is set. The number of bytes received in the endpoint's USB_BCR register is equal to zero.
3. The microcontroller must clear RX_DATA_BK0.

Figure 173. Data IN Followed by Status OUT Transfer



Stall Handshake

A stall handshake can be used in one of two distinct occasions. (For more information on the stall handshake, refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0.*)

- A functional stall is used when the halt feature associated with the endpoint is set. (Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the halt feature.)
- To abort the current request, a protocol stall is used, but uniquely with control transfer.

The following procedure generates a stall packet:

1. The microcontroller sets the FORCESTALL flag in the USB_CSRx endpoint's register.
2. The host receives the stall packet.
3. The microcontroller is notified that the device has sent the stall by polling the STALLSENT to be set. An endpoint interrupt is pending while STALLSENT is set. The microcontroller must clear STALLSENT to clear the interrupt.

When a setup transaction is received after a stall handshake, STALLSENT must be cleared in order to prevent interrupts due to STALLSENT being set.

Figure 174. Stall Handshake (Data IN Transfer)

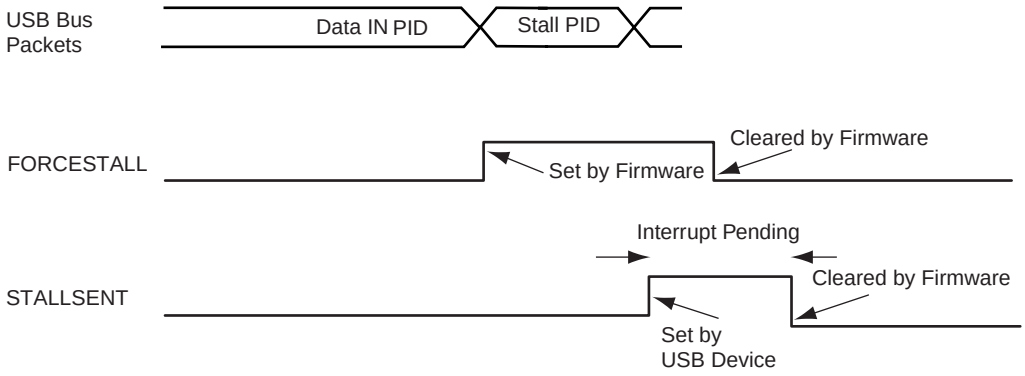
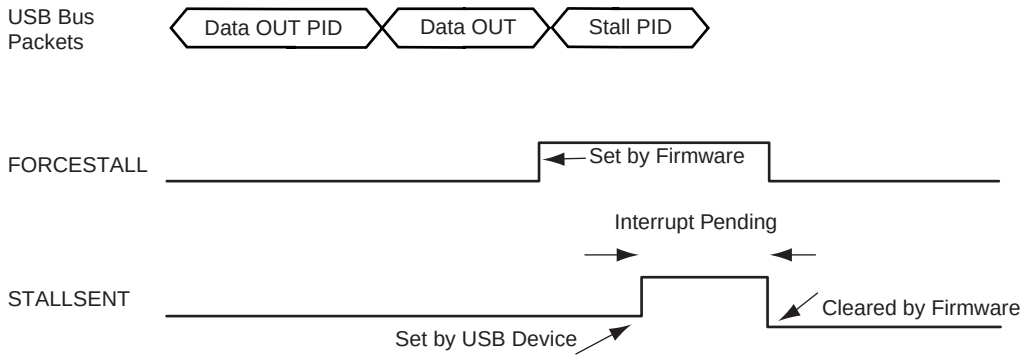


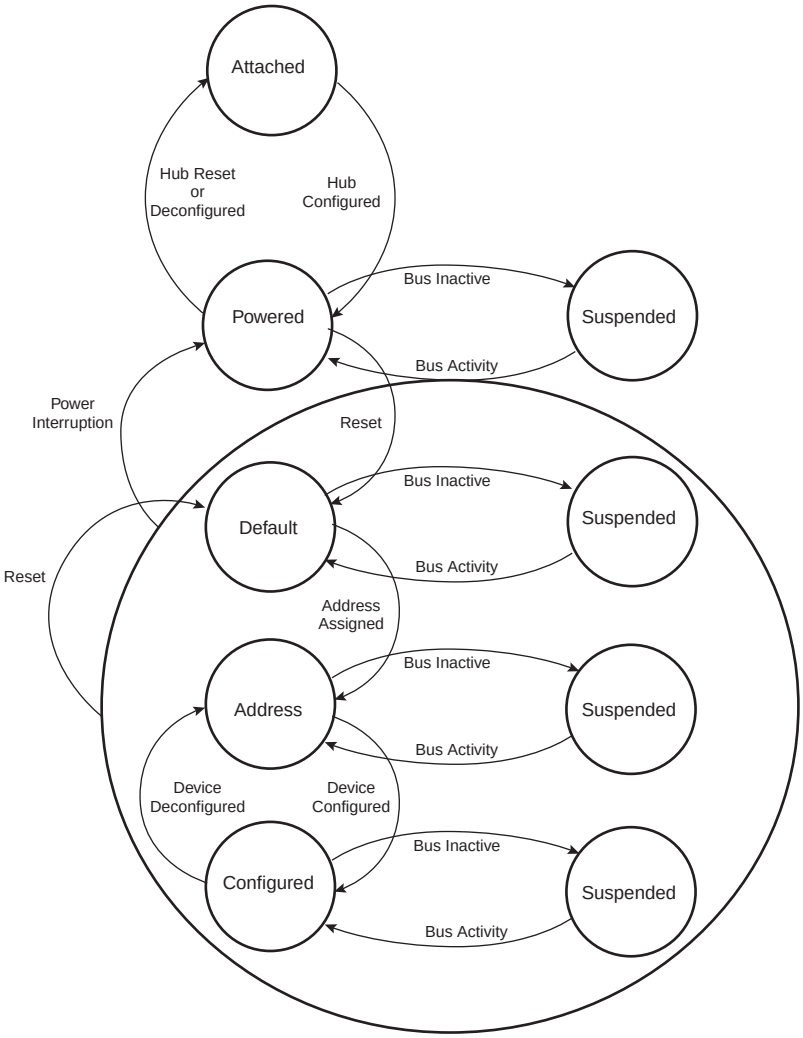
Figure 175. Stall Handshake (Data OUT Transfer)



Controlling Device States

A USB device has several possible states. Please refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*.

Figure 176. USB Device State Diagram



Movement from one state to another depends on the USB bus state or on standard requests sent through control transactions via the default endpoint (endpoint 0).

After a period of bus inactivity, the UDP device enters Suspend Mode. Accepting Suspend/Resume requests from the USB host is mandatory. Constraints in Suspend Mode are very strict for bus-powered applications; devices may not consume more than 500 uA on the USB bus.

While in Suspend Mode, the host may wake up a device by sending a resume signal (bus activity) or a USB device may send a wake-up request to the host, e.g., waking up a PC by moving a USB mouse.

The wake-up feature is not mandatory for all devices and must be negotiated with the host.

From Powered State to Default State

After its connection to a USB host, the USB device waits for an end-of-bus reset. The USB host stops driving a reset state once it has detected the device's pull-up on DP. The unmasked flag ENDBURST is set in the register UDP_ISR and an interrupt is triggered. The UDP software enables the default endpoint, setting the EPEDS flag in the UDP_CSR[0] register and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the UDP_IER register. The enumeration then begins by a control transfer.

From Default State to Address State

After a set address standard device request, the USB host peripheral enters the address state. Before this, it achieves the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP_CSR[0] register has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the UDP_GLB_STATE, sets its new address, and sets the FEN bit in the UDP_FADDR register.

From Address State to Configured State

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP_CSRx registers and, optionally, enabling corresponding interrupts in the UDP_IER register.

Enabling Suspend

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP_ISR register is set. This triggers an interrupt if the corresponding bit is set in the UDP_IMR register.

This flag is cleared by writing to the UDP_ICR register. Then the device enters Suspend Mode. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle Mode. It may also switch off other devices on the board.

The USB device peripheral clocks may be switched off. However, the transceiver and the USB peripheral must not be switched off, otherwise the resume is not detected.

Receiving a Host Resume

In suspend mode, the USB transceiver and the USB peripheral must be powered to detect the RESUME. However, the USB device peripheral may not be clocked as the WAKEUP signal is asynchronous.

Once the resume is detected on the bus, the signal WAKEUP in the UDP_ISR is set. It may generate an interrupt if the corresponding bit in the UDP_IMR register is set. This interrupt may be used to wake-up the core, enable PLL and main oscillators and configure clocks. The WAKEUP bit must be cleared as soon as possible by setting WAKEUP in the UDP_ICR register.

Sending an External Resume

The External Resume is negotiated with the host and enabled by setting the ESR bit in the USB_GLB_STATE. An asynchronous event on the ext_resume_pin of the peripheral generates a WAKEUP interrupt. On early versions of the USP peripheral, the K-state on the USB line is generated immediately. This means that the USB device must be able to answer to the host very quickly. On recent versions, the software sets the RMWUPE bit in the UDP_GLB_STATE register once it is ready to communicate with the host. The K-state on the bus is then generated.

The WAKEUP bit must be cleared as soon as possible by setting WAKEUP in the UDP_ICR register.

USB Device Port (UDP) User Interface

Table 71. USB Device Port Memory Map

Offset	Register	Name	Access	Reset State
0x000	Frame Number Register	USB_FRM_NUM	Read	0x0000_0000
0x004	Global State Register	USB_GLB_STAT	Read/write	0x0000_0010
0x008	Function Address Register	USB_FADDR	Read/write	0x0000_0100
0x00C	Reserved	–	–	–
0x010	Interrupt Enable Register	USB_IER	Write	
0x014	Interrupt Disable Register	USB_IDR	Write	
0x018	Interrupt Mask Register	USB_IMR	Read	0x0000_1200
0x01C	Interrupt Status Register	USB_ISR	Read	0x0000_0000
0x020	Interrupt Clear Register	USB_ICR	Write	
0x024	Reserved	–	–	–
0x028	Reset Endpoint Register	USB_RST_EP	Read/write	
0x02C	Reserved	–	–	–
0x030	Endpoint 0 Control and Status Register	USB _CSR0	Read/write	0x0000_0000
0x034	Endpoint 1 Control and Status Register	USB _CSR1	Read/write	0x0000_0000
0x038	Endpoint 2 Control and Status Register	USB _CSR2	Read/write	0x0000_0000
0x03C	Endpoint 3 Control and Status Register	USB _CSR3	Read/write	0x0000_0000
0x040	Endpoint 4 Control and Status Register	USB _CSR4	Read/write	0x0000_0000
0x044	Endpoint 5 Control and Status Register	USB _CSR5	Read/write	0x0000_0000
0x048	Endpoint 6 Control and Status Register	USB _CSR6	Read/write	0x0000_0000
0x04C	Endpoint 7 Control and Status Register	USB _CSR7	Read/write	0x0000_0000
0x050	Endpoint 0 FIFO Data Register	USB_FDR0	Read/write	0x0000_0000
0x054	Endpoint 1 FIFO Data Register	USB_FDR1	Read/write	0x0000_0000
0x058	Endpoint 2 FIFO Data Register	USB_FDR2	Read/write	0x0000_0000
0x05C	Endpoint 3 FIFO Data Register	USB_FDR3	Read/write	0x0000_0000
0x060	Endpoint 4 FIFO Data Register	USB_FDR4	Read/write	0x0000_0000
0x064	Endpoint 5 FIFO Data Register	USB_FDR5	Read/write	0x0000_0000
0x068	Endpoint 6 FIFO Data Register	USB_FDR6	Read/write	0x0000_0000
0x06C	Endpoint 7 FIFO Data Register	USB_FDR7	Read/write	0x0000_0000
0x070	Reserved	–	–	–
0x074	Reserved	–	–	–

USB Frame Number Register

Register Name: USB_FRM_NUM

Access Type: Read-only

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

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

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

7	6	5	4	3	2	1	0
FRM_NUM							

- **FRM_NUM[10:0]: Frame Number as Defined in the Packet Field Formats**

This 11-bit value is incremented by the host on a per frame basis. This value is updated at each start of frame.

Value Updated at the SOF_EOP (Start of Frame End of Packet).

- **FRM_ERR: Frame Error**

This bit is set at SOF_EOP when the SOF packet is received containing an error.

This bit is reset upon receipt of SOF_PID.

- **FRM_OK: Frame OK**

This bit is set at SOF_EOP when the SOF packet is received without any error.

This bit is reset upon receipt of SOF_PID (Packet Identification).

In the Interrupt Status Register, the SOF interrupt is updated upon receiving SOF_PID. This bit is set without waiting for EOP.

Note: In the 8-bit Register Interface, FRM_OK is bit 4 of FRM_NUM_H and FRM_ERR is bit 3 of FRM_NUM_L.

USB Global State Register

Register Name: USB_GLB_STAT

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	RMWUPE	RSMINPR	ESR	CONFG	FADDEN

This register is used to get and set the device state as specified in Chapter 9 of the *USB Serial Bus Specification, Rev.2.0*.

- **FADDEN: Function Address Enable**

Read:

0 = Device is not in address state.

1 = Device is in address state.

Write:

0 = No effect, only a reset can bring back a device to the default state.

1 = Set device in address state. This occurs after a successful Set Address request. Beforehand, the USB_FADDR register must have been initialized with Set Address parameters. Set Address must complete the Status Stage before setting FADDEN. Please refer to chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* to get more details.

- **CONFG: Configured**

Read:

0 = Device is not in configured state.

1 = Device is in configured state.

Write:

0 = Set device in a nonconfigured state

1 = Set device in configured state.

The device is set in configured state when it is in address state and receives a successful Set Configuration request. Please refer to Chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* to get more details.

- **ESR: Enable Send Resume**

0 = Disable the Remote Wake Up sequence.

1 = Remote Wake Up can be processed and the pin send_resume is enabled.

- **RSMINPR: A Resume Has Been Sent to the Host**

Read:

0 = No effect.

1 = A Resume has been received from the host during Remote Wake Up feature.

- **RMWUPE: Remote Wake Up Enable**

0 = Must be cleared after receiving any HOST packet or SOF interrupt.

1 = Enables the K-state on the USB cable if ESR is enabled.

USB Function Address Register

Register Name: USB_FADDR

Access Type: Read/Write

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

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

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

7	6	5	4	3	2	1	0
–	FADD						

- **FADD[6:0]: Function Address Value**

The Function Address Value must be programmed by firmware once the device receives a set address request from the host, and has achieved the status stage of the no-data control sequence. Please refer to the *Universal Serial Bus Specification, Rev. 2.0* to get more information. After power up, or reset, the function address value is set to 0.

- **FEN: Function Enable**

Read:

0 = Function endpoint disabled.

1 = Function endpoint enabled.

Write:

0 = Disable function endpoint.

1 = Default value.

The Function Enable bit (FEN) allows the microcontroller to enable or disable the function endpoints. The microcontroller sets this bit after receipt of a reset from the host. Once this bit is set, the USB device is able to accept and transfer data packets from and to the host.

USB Interrupt Enable Register

Register Name: USB_IER

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Enable Endpoint 0 Interrupt**
- **EP1INT: Enable Endpoint 1 Interrupt**
- **EP2INT: Enable Endpoint 2Interrupt**
- **EP3INT: Enable Endpoint 3 Interrupt**
- **EP4INT: Enable Endpoint 4 Interrupt**
- **EP5INT: Enable Endpoint 5 Interrupt**
- **EP6INT: Enable Endpoint 6 Interrupt**
- **EP7INT: Enable Endpoint 7 Interrupt**

0 = No effect.

1 = Enable corresponding Endpoint Interrupt.

- **RXSUSP: Enable USB Suspend Interrupt**

0 = No effect.

1 = Enable USB Suspend Interrupt.

- **RXRSM: Enable USB Resume Interrupt**

0 = No effect.

1 = Enable USB Resume Interrupt.

- **EXTRSM: Enable External Resume Interrupt**

0 = No effect.

1 = Enable External Resume Interrupt.

- **SOFINT: Enable Start Of Frame Interrupt**

0 = No effect.

1 = Enable Start Of Frame Interrupt.

- **WAKEUP: Enable USB bus Wakeup Interrupt**

0 = No effect.

1 = Enable USB bus Interrupt.

USB Interrupt Disable Register

Register Name: USB_IDR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Disable Endpoint 0 Interrupt**
- **EP1INT: Disable Endpoint 1 Interrupt**
- **EP2INT: Disable Endpoint 2 Interrupt**
- **EP3INT: Disable Endpoint 3 Interrupt**
- **EP4INT: Disable Endpoint 4 Interrupt**
- **EP5INT: Disable Endpoint 5 Interrupt**
- **EP6INT: Disable Endpoint 6 Interrupt**
- **EP7INT: Disable Endpoint 7 Interrupt**

0 = No effect.

1 = Disable corresponding Endpoint Interrupt.

- **RXSUSP: Disable USB Suspend Interrupt**

0 = No effect.

1 = Disable USB Suspend Interrupt.

- **RXRSM: Disable USB Resume Interrupt**

0 = No effect.

1 = Disable USB Resume Interrupt.

- **EXTRSM: Disable External Resume Interrupt**

0 = No effect.

1 = Disable External Resume Interrupt.

- **SOFINT: Disable Start Of Frame Interrupt**

0 = No effect.

1 = Disable Start Of Frame Interrupt

- **WAKEUP: Disable USB Bus Interrupt**

0 = No effect.

1 = Disable USB Bus Wakeup Interrupt.

USB Interrupt Mask Register

Register Name: USB_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
–	–	WAKEUP	–	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Mask Endpoint 0 Interrupt**
- **EP1INT: Mask Endpoint 1 Interrupt**
- **EP2INT: Mask Endpoint 2 Interrupt**
- **EP3INT: Mask Endpoint 3 Interrupt**
- **EP4INT: Mask Endpoint 4 Interrupt**
- **EP5INT: Mask Endpoint 5 Interrupt**
- **EP6INT: Mask Endpoint 6 Interrupt**
- **EP7INT: Mask Endpoint 7 Interrupt**

0 = Corresponding Endpoint Interrupt is disabled.

1 = Corresponding Endpoint Interrupt is enabled.

- **RXSUSP: Mask USB Suspend Interrupt**

0 = USB Suspend Interrupt is disabled.

1 = USB Suspend Interrupt is enabled.

- **RXRSM: Mask USB Resume Interrupt.**

0 = USB Resume Interrupt is disabled.

1 = USB Resume Interrupt is enabled.

- **EXTRSM: Mask External Resume Interrupt**

0 = External Resume Interrupt is disabled.

1 = External Resume Interrupt is enabled.

- **SOFINT: Mask Start Of Frame Interrupt**

0 = Start of Frame Interrupt is disabled.

1 = Start of Frame Interrupt is enabled.

- **WAKEUP: USB Bus WAKEUP Interrupt**

0 = USB Bus Wakeup Interrupt is disabled.

1 = USB Bus Wakeup Interrupt is enabled.

Note: When the USB block is in suspend mode, the application may power down the USB logic. In this case, any USB HOST resume request that is made must be taken into account and, thus, the reset value of the RXRSM bit of the register USB_IMR is enabled.

USB Interrupt Status Register

Register Name: USB_ISR

Access Type: Read -only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBUSRES	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Endpoint 0 Interrupt Status**

0 = No Endpoint0 Interrupt pending.

1 = Endpoint0 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR0:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP0INT is a sticky bit. Interrupt remains valid until EP0INT is cleared by writing in the corresponding USB_CSR0 bit.

- **EP1INT: Endpoint 1 Interrupt Status**

0 = No Endpoint1 Interrupt pending.

1 = Endpoint1 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR1:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP1INT is a sticky bit. Interrupt remains valid until EP1INT is cleared by writing in the corresponding USB_CSR1 bit.

- **EP2INT: Endpoint 2 Interrupt Status**

0 = No Endpoint2 Interrupt pending.

1 = Endpoint2 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR2:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP2INT is a sticky bit. Interrupt remains valid until EP2INT is cleared by writing in the corresponding USB_CSR2 bit.

- **EP3INT: Endpoint 3 Interrupt Status**

0 = No Endpoint3 Interrupt pending.

1 = Endpoint3 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR3:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP3INT is a sticky bit. Interrupt remains valid until EP3INT is cleared by writing in the corresponding USB_CSR3 bit.

- **EP4INT: Endpoint 4 Interrupt Status**

0 = No Endpoint4 Interrupt pending.

1 = Endpoint4 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR4:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP4INT is a sticky bit. Interrupt remains valid until EP4INT is cleared by writing in the corresponding USB_CSR4 bit.

- **EP5INT: Endpoint 5 Interrupt Status**

0 = No Endpoint5 Interrupt pending.

1 = Endpoint5 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR5:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP5INT is a sticky bit. Interrupt remains valid until EP5INT is cleared by writing in the corresponding USB_CSR5 bit.

- **EP6INT: Endpoint 6 Interrupt Status**

0 = No Endpoint6 Interrupt pending.

1 = Endpoint6 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR6:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP6INT is a sticky bit. Interrupt remains valid until EP6INT is cleared by writing in the corresponding USB_CSR6 bit.

- **EP7INT: Endpoint 7 Interrupt Status**

0 = No Endpoint7 Interrupt pending.

1 = Endpoint7 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading USB_CSR7:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP7INT is a sticky bit. Interrupt remains valid until EP7INT is cleared by writing in the corresponding USB_CSR7 bit.

- **RXSUSP: USB Suspend Interrupt Status**

0 = No USB Suspend Interrupt pending.

1 = USB Suspend Interrupt has been raised.

The USB device sets this bit when it detects no activity for 3ms. The USB device enters Suspend mode.

- **RXRSM: USB Resume Interrupt Status**

0 = No USB Resume Interrupt pending.

1 =USB Resume Interrupt has been raised.

The USB device sets this bit when a USB resume signal is detected at its port.

- **EXTRSM: External Resume Interrupt Status**

0 = No External Resume Interrupt pending.

1 = External Resume Interrupt has been raised.

This interrupt is raised when, in suspend mode, an asynchronous rising edge on the send_resume is detected.

If RMWUPE = 1, a resume state is sent in the USB bus.

- **SOFINT: Start of Frame Interrupt Status**

0 = No Start of Frame Interrupt pending.

1 = Start of Frame Interrupt has been raised.

This interrupt is raised each time a SOF token has been detected. It can be used as a synchronization signal by using isochronous endpoints.

- **ENDBUSRES: End of BUS Reset Interrupt Status**

0 = No End of Bus Reset Interrupt pending.

1 = End of Bus Reset Interrupt has been raised.

This interrupt is raised at the end of a USB reset sequence. The USB device must prepare to receive requests on the end-point 0. The host starts the enumeration, then performs the configuration.

- **WAKEUP: USB Resume Interrupt Status**

0 = No Wakeup Interrupt pending.

1 = A Wakeup Interrupt (USB Host Sent a RESUME or RESET) occurred since the last clear.

USB Interrupt Clear Register

Register Name: USB_ICR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBURST	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **RXSUSP: Clear USB Suspend Interrupt**

0 = No effect.

1 = Clear USB Suspend Interrupt.

- **RXRSM: Clear USB Resume Interrupt**

0 = No effect.

1 = Clear USB Resume Interrupt.

- **EXTRSM: Clear External Resume Interrupt**

0 = No effect.

1 = Clear External Resume Interrupt.

- **SOFINT: Clear Start Of Frame Interrupt**

0 = No effect.

1 = Clear Start Of Frame Interrupt.

- **ENDBURST: Clear End of Bus Reset Interrupt**

0 = No effect.

1 = Clear Start Of Frame Interrupt.

- **WAKEUP: Clear Wakeup Interrupt**

0 = No effect.

1 = Clear Wakeup Interrupt.

USB Reset Endpoint Register

Register Name: USB_RST_EP

Access Type: Read/write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0

- **EP0: Reset Endpoint 0**
- **EP1: Reset Endpoint 1**
- **EP2: Reset Endpoint 2**
- **EP3: Reset Endpoint 3**
- **EP4: Reset Endpoint 4**
- **EP5: Reset Endpoint 5**
- **EP6: Reset Endpoint 6**
- **EP7: Reset Endpoint 7**

This flag is used to reset the FIFO associated with the endpoint and the bit RXBYTECOUNT in the register UDP_CSRx.It also resets the data toggle to DATA0. It is useful after removing a HALT condition on a BULK endpoint. Refer to Chapter 5.8.5 in the *USB Serial Bus Specification, Rev. 2.0*.

Warning: This flag must be cleared at the end of the reset. It does not clear USB_CSRx flags.

0 = No reset.

1 = Forces the corresponding endpoint FIF0 pointers to 0, therefore RXBYTECNT field is read at 0 in USB_CSRx register.

USB Endpoint Control and Status Register

Register Name: USB_CSRx [x = 0. 7]

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
RXBYTECNT							

15	14	13	12	11	10	9	8
EPEDS	–	–	–	DTGLE	EPTYPE		

7	6	5	4	3	2	1	0
DIR	RX_DATA_ BK1	FORCE STALL	TXPKTRDY	STALLSENT ISOERROR	RXSETUP	RX_DATA_ BK0	TXCOMP

- **TXCOMP: Generates an IN packet with data previously written in the DPR**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware)

0 = Clear the flag, clear the interrupt.

1 = No effect.

Read (Set by the USB peripheral)

0 = Data IN transaction has not been acknowledged by the Host.

1 = Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX_DATA_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware)

0 = Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1 = No effect.

Read (Set by the USB peripheral)

0 = No data packet has been received in the FIFO's Bank 0

1 = A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the USB_FDRx register. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX_DATA_BK0.

- **RXSETUP: Sends STALL to the Host (Control endpoints)**

This flag generates an interrupt while it is set to one.

Read

0 = No setup packet available.

1 = A setup data packet has been sent by the host and is available in the FIFO.

Write

0 = Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1 = No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the USB_FDRx register to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transactions is not accepted while RXSETUP is set.

- **STALLSENT: Stall sent (Control, Bulk Interrupt endpoints)/ ISOERROR (Isochronous endpoints)**

This flag generates an interrupt while it is set to one.

STALLSENT: this ends a STALL handshake

Read

0 = the host has not acknowledged a STALL.

1 = host has acknowledge the stall.

Write

0 = reset the STALLSENT flag, clear the interrupt.

1 = No effect.

This is mandatory for the device firmware to clear this flag. Otherwise the interrupt remains.

Please refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* to get more information on the STALL handshake.

ISOERROR: a CRC error has been detected in an isochronous transfer

Read

0 = No error in the previous isochronous transfer.

1 = CRC error has been detected, data available in the FIFO are corrupted.

Write

0 = reset the ISOERROR flag, clear the interrupt.

1 = No effect.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read

0 = Data values can be written in the FIFO.

1 = Data values can not be written in the FIFO.

Write

0 = No effect.

1 = A new data payload is has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the USB_FDRx register. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous endpoints)**

Write-only

0 = No effect.

1 = Send STALL to the host.

Please refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* to get more information on the STALL handshake.

Control endpoints: during the data stage and status stage, this indicates that the microcontroller can not complete the request.

Bulk and interrupt endpoints: notify the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX_DATA_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware)

0 = Notify USB device that data have been read in the FIFO's Bank 1.

1 = No effect.

Read (Set by the USB peripheral)

0 = No data packet has been received in the FIFO's Bank 1.

1 = A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through USB_FDRx register. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX_DATA_BK1.

- **DIR: Transfer Direction (only available for control endpoints)**

Read/Write

0 = Allow Data OUT transactions in the control data stage.

1 = Enable Data IN transactions in the control data stage.

Please refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* to get more information on the control data stage.

This bit must be set before USB_CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type**

Read/Write

000	Control
001	Isochronous OUT
101	Isochronous IN
010	Bulk OUT
110	Bulk IN
011	Interrupt OUT
111	Interrupt IN

- **DTGLE: Data Toggle**

Read-only

0 = Identifies DATA0 packet.

1 = Identifies DATA1 packet.

Please refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* to get more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read

0 = Endpoint disabled.

1 = Endpoint enabled.

Write

0 = Disable endpoint.

1 = Enable endpoint.

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO**

Read-only.

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the USB_FDRx register.

USB FIFO Data Register

Register Name: USB_FDRx [x = 0. 7]

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
FIFO_DATA							

- **FIFO_DATA[7:0]: FIFO Data Value**

The microcontroller can push or pop values in the FIFO through this register.

RXBYTECNT in the corresponding USB_CSRx register is the number of bytes to be read from the FIFO (sent by the host).

The maximum number of bytes to write is fixed by the Max Packet Size in the Standard Endpoint Descriptor. It can not be more than the physical memory size associated to the endpoint. Please refer to the *Universal Serial Bus Specification, Rev. 2.0* to get more information.

DC Characteristics

Absolute Maximum Ratings

Table 72. 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 +3.6V
Maximum Operating Voltage (V_{DDCORE}, V_{DDPLL} and V_{DDOSC}) 1.95V
Maximum Operating Voltage (V_{DDIO})..... 3.6V
DC Output Current..... 8 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°C to 85°C, unless otherwise specified and are certified for a junction temperature up to T_J = 100°C.

Table 73. DC Characteristics

Symbol	Parameter	Conditions		Min	Typ	Max	Units
V_{DDCORE}	DC Supply Core			1.65		1.95	V
V_{DDOSC}	DC Supply Oscillator			1.65		1.95	V
V_{DDPLL}	DC Supply PLL			1.65		1.95	V
V_{DDIO}	DC Supply I/Os			V_{DDCORE}		V_{DDCORE} + 1.5 or 3.6	V
V_{IL}	Input Low-level Voltage	Pin Group 1[1] and Pin Group 4[4 Pin Group 2[2] and Pin Group 3[3]		-0.3 -0.3		0.3xV_{DDIO} 0.8	V
V_{IH}	Input High-level Voltage	Pin Group 1[1] and Pin Group 4[4] Pin Group 2[2] and Pin Group 3[3]		0.7xV_{DDIO} 2.0		V_{DDIO}+0.3 V_{DDIO}+0.3	V
V_{OL}	Output Low-level Voltage	Pin Group 1[1] I_{OL} = 8 mA[7]				0.4	V
		Pin Group 5[5] I_{OL} = 4 mA[7] I_{OL} = 8 mA[7]				0.2 0.4	
V_{OH}	Output High-level Voltage	Pin Group 1[1] I_{OH} = 8 mA[7]		V_{DDIO}-0.4			V
		Pin Group 5[5] I_{OH} = 4 mA[7] I_{OH} = 8 mA[7]		V_{DDIO}-0.2 V_{DDIO}-0.4			
I_{LEAK}	Input Leakage Current	Pullup resistors disabled				1	µA
I_{PULL}	Input Pull-up Current	Pin Group 1[1] V_{DD} = 3.0V[6], V_{IN} = 0 V_{DD} = 3.6V[6], V_{IN} = 0		9.6		26.6	µA
		Pin Group 3[3] V_{DD} = 3.3V[6], V_{IN} = 0		122.7		339	µA
		Pin Group 4[4] V_{DD} = 3.0V[6], V_{IN} = 0 V_{DD} = 3.6V[6], V_{IN} = 0		157.8		363	µA
C_{IN}	Input Capacitance	100-LQFP Package				8.3	pF
I_{SC}	Static Current	On V_{DDCORE} = 2V, MCK = 0 Hz	T_A = 25°C		15	45	µA
		All inputs driven TMS, TDI, TCK, NRST = 1	T_A = 85°C		130	340	

4. Pin Group 1: PIOA and PIOB
5. Pin Group 2: NRST
6. Pin Group 3: JTAGSEL/TCK/TMS/TST
7. Pin Group 4: TDI
8. Pin Group 5: TDO
9. V_{DD} is applicable to V_{DDIO},V_{DDPLL} and V_{DDOSC}
10. I_O= Output Current

Clocks Characteristics

These parameters are given in the following conditions:

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

The Temperature Derating Factor described in “Applicable Conditions and Derating Data” on page 439, section “Temperature Derating Factor” on page 440 and V_{DDCORE} Voltage Derating Factor described in “Applicable Conditions and Derating Data” on page 439, section “VDDCORE Voltage Derating Factor” on page 440 are both applicable to these characteristics.

Processor Clock Characteristics

Table 74. Processor Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPPCK})$	Processor Clock Frequency			90	MHz
t_{CPPCK}	Processor Clock Period		11.1		ns

Master Clock Characteristics

Table 75. Master Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPMCK})$	Master Clock Frequency			70.0	MHz
t_{CPMCK}	Master Clock Period		14.3		ns
t_{CHMCK}	Master Clock High Half-period		7.1		ns
t_{CLMCK}	Master Clock Low Half-period		7.1		ns

XIN Clock Characteristics

Table 76. 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 x t_{CPXIN}	0.6 x t_{CPXIN}	
t_{CLXIN}	XIN Clock Low Half-period		0.4 x t_{CPXIN}	0.6 x t_{CPXIN}	
C_{IN}	XIN Input Capacitance	(1)		25	pF
R_{IN}	XIN Pulldown Resistor	(1)		500	kΩ

Notes: 1. These characteristics apply only when the Main Oscillator is in Bypass Mode (i.e., when MOSCEN = 0 in the CKGR_MOR register, refer to “PMC Clock Generator Main Oscillator Register” on page 146).

Power Consumption

The values in Table 77 and Table 78 are measured values on the AT91RM3400DK Evaluation Board with operating conditions as follows:

- V_{DDIO} = 3.3V
- V_{DDCORE} = V_{DDPLL} = V_{DDOSC} = 1.8V
- T_A = 25°C
- MCK = 48 MHz
- PCK = 48 MHz
- SLCK = 32,768 Hz

These figures represent the power consumption measured on the V_{DDCORE} power supply.

Table 77. Power Consumption for PMC Modes[1]

Mode	Conditions	Consumption	Unit
Active	ARM Core clock enabled. All peripheral clocks activated.	34.1	mA
Normal	ARM Core clock enabled. All peripheral clocks deactivated.	19.4	
Idle	ARM Core clock disabled and waiting for the next interrupt. All peripheral clocks deactivated.	5.1	
Slow Clock	Main oscillator and PLLs are switched off. Processor and all peripherals run at slow clock.	0.6	
Standby	Combination of Idle and Slow Clock Modes.	0.5	

Note: 1. Code in internal SRAM.

Table 78. Power Consumption by Peripheral[1]

Peripheral	Consumption	Unit
PIO Controller	0.4	mA
USART	0.9	
MCI	1.2	
UDP	1.0	
TWI	0.2	
SPI	0.9	
SSC	1.1	
Timer Counter Channels	0.2	
PMC		
PLL[2]	4	mA
Slow Clock Oscillator[3]	1.3	µA
Main Oscillator[3]	550	µA

Notes: 1. Code in internal SRAM.
2. Power consumption on the V_{DDPLL} power supply.
3. Power consumption on the V_{DDOSC} power supply.

Crystal Oscillators Characteristics

32 kHz Oscillator Characteristics

Table 79. 32 kHz Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CP32KHz})$	Crystal Oscillator Frequency			32.768		kHz
	Duty Cycle	Measured at the PCK output pin	40	50	60	%
t_{ST}	Startup Time	V_{DDOSC} = 1.2 to 2V R_s = 50 kΩ, C_L =12.5 pF(1)			900	ms

Notes: 1. R_s is the equivalent series resistance, C_L is the equivalent load capacitance

Main Oscillator Characteristics

Table 80. 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(1) $1/(t_{CPMAIN})$ = 3 MHz C_S = 7 pF(1) $1/(t_{CPMAIN})$ = 16 MHz C_S = 7 pF(1) $1/(t_{CPMAIN})$ = 20 MHz			14.5 1.4 1	ms

Notes: 1. C_S is the shunt capacitance

PLL Characteristics

Table 81. Phase Lock Loop Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
F_{OUT}	Output Frequency		20		100	MHz
F_{IN}	Input Frequency		1		32	MHz
K_O	VCO Gain		120	190	300	MHz/V
I_P	Pump Current		36	44	60	µA

Transceiver Characteristics

Electrical Characteristics

Table 82. Electrical Parameters

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Input Levels						
V_{IL}	Low Level				0.8	V
V_{IH}	High Level		2.0			V
V_{DI}	Differential Input Sensivity	|(D+) - (D-)|	0.2			V
V_{CM}	Differential Input Common Mode Range		0.8		2.5	V
C_{IN}	Transceiver capacitance	Capacitance to ground on each line			9.18	pF
I	Hi-Z State Data Line Leakage	$0V < V_{IN} < 3.3V$	-10		10	µA
R_{EXT}	Recommended External USB Series Resistor	In series with each USB pin with ±5%		27		Ω
Output Levels						
V_{OL}	Low Level Output	Measured with R_L of 1.425 kOhm tied to 3.6V	0.0		0.3	V
V_{OH}	High Level Output	Measured with R_L of 14.25 kOhm tied to GND	2.8		3.6	V
V_{CRS}	Output Signal Crossover Voltage	Measure conditions described in Figure 177	1.3		2.0	V

Switching Characteristics

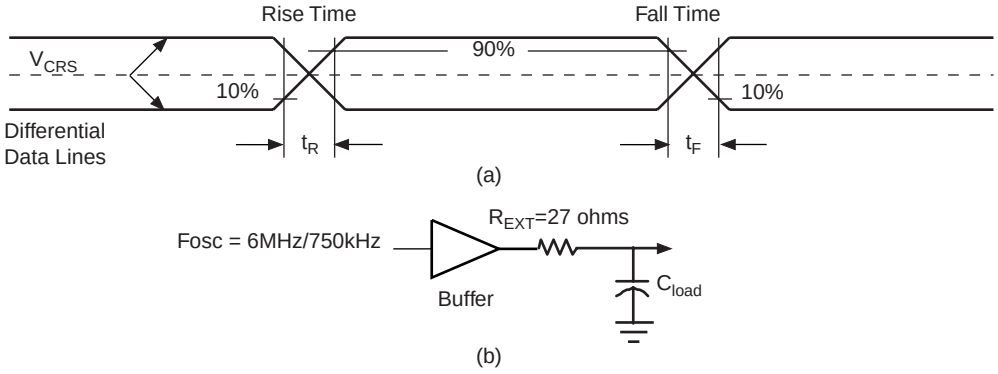
Table 83. In Low Speed

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t_{FR}	Transition Rise Time	C_{LOAD} = 400 pF	75		300	ns
t_{FE}	Transition Fall Time	C_{LOAD} = 400 pF	75		300	ns
t_{FRFM}	Rise/Fall Time Matching	C_{LOAD} = 400 pF	80		125	%

Table 84. In Full Speed

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t_{FR}	Transition Rise Time	C_{LOAD} = 50 pF	4		20	ns
t_{FE}	Transition Fall Time	C_{LOAD} = 50 pF	4		20	ns
t_{FRFM}	Rise/Fall Time Matching		90		111.11	%

Figure 177. USB Data Signal Rise and Fall Times



AC Characteristics

Applicable Conditions and Derating Data

Conditions and Timings Computation

The 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 x V_{DDIO}).
- The input level is (0.3 x V_{DDIO}) for a low-level detection and is (0.7 x V_{DDIO}) 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 timingfor other conditions, the following equation should be used:

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

where:

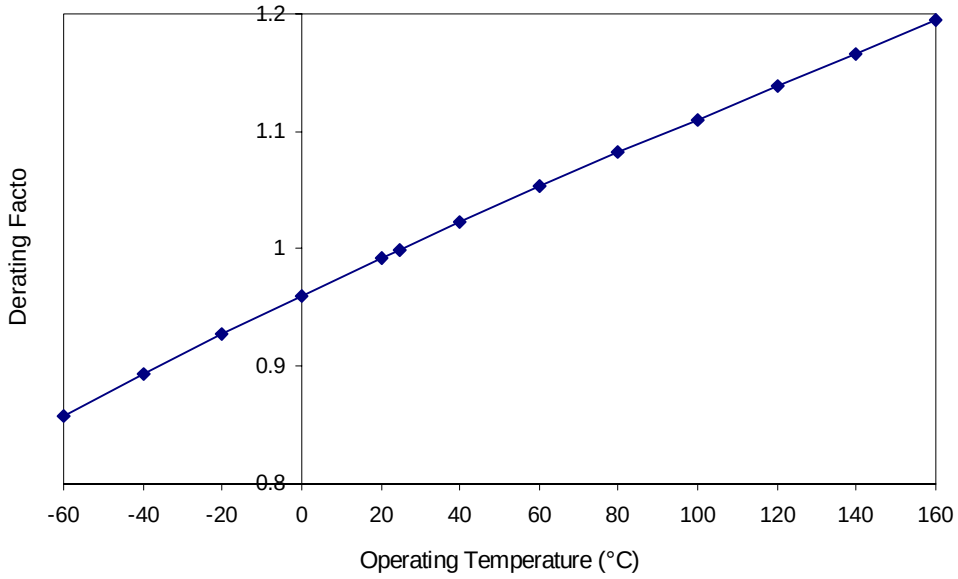
- δ_{T° is the derating factor in temperature given in Figure 178 on page 440.
- $\delta_{VDDCORE}$ is the derating factor for the Core Power Supply given in Figure 179 on page 440.
- $t_{DATASHEET}$ is the minimum or maximum timing value given in this datasheet for a load capacitance of 0 pF.
- δ_{VDDIO} is the derating factor for the IO Power Supply given in Figure 180 on page 441.
- C_{SIGNAL} is the capacitance load on the considered output pin(1).
- $\delta_{CSIGNAL}$ 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: 1. The user must take into account the package capacitance load contribution (C_{IN}) described in Table 73 on page 432.

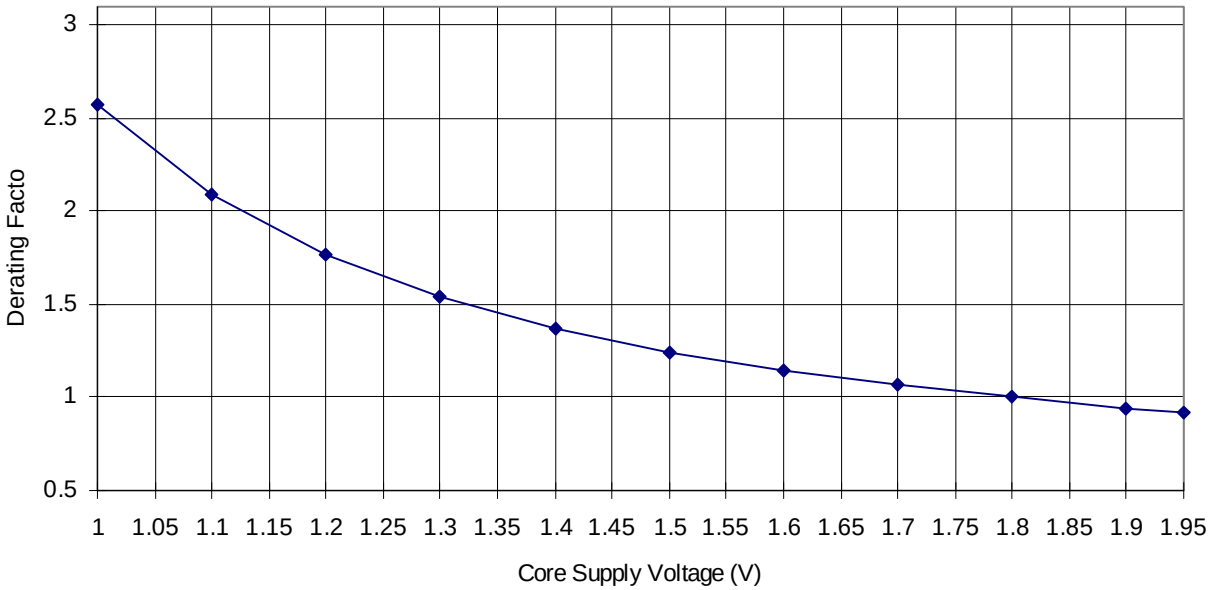
Temperature Derating Factor

Figure 178. Derating Curve for Different Operating Temperatures



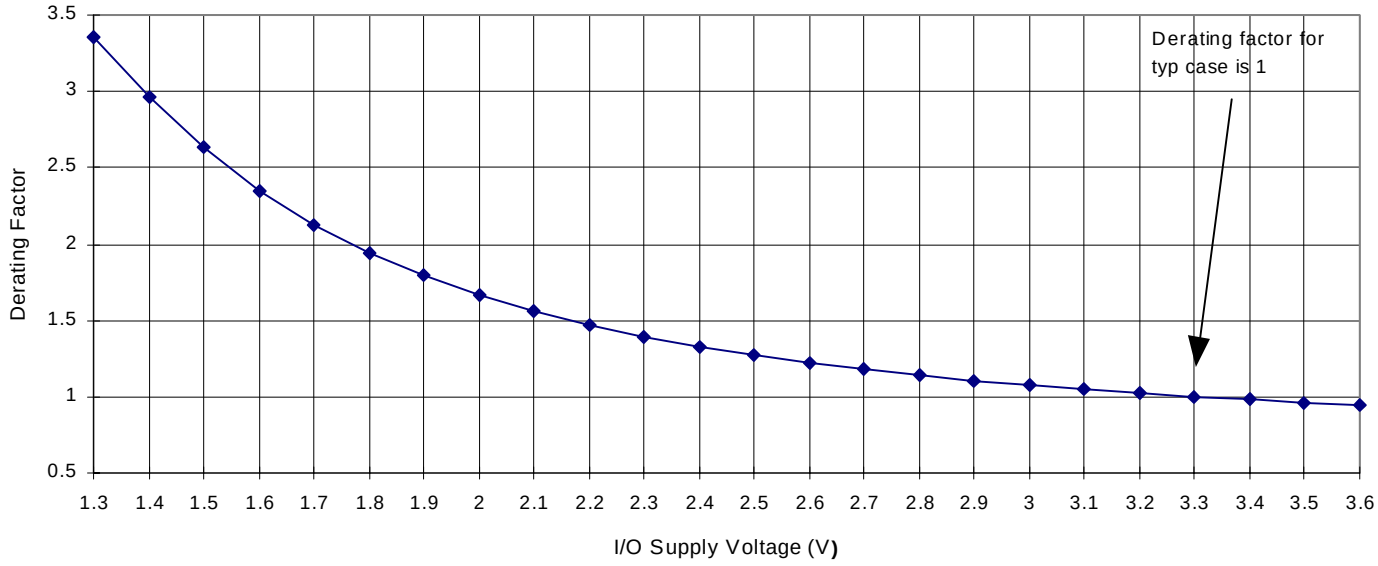
V_{DDCORE} Voltage Derating Factor

Figure 179. Derating Curve for Different Core Supply Voltages



V_{DDIO} Voltage Derating Factor

Figure 180. Derating Curve for Different IO Supply Voltages



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

JTAG/ICE Timings

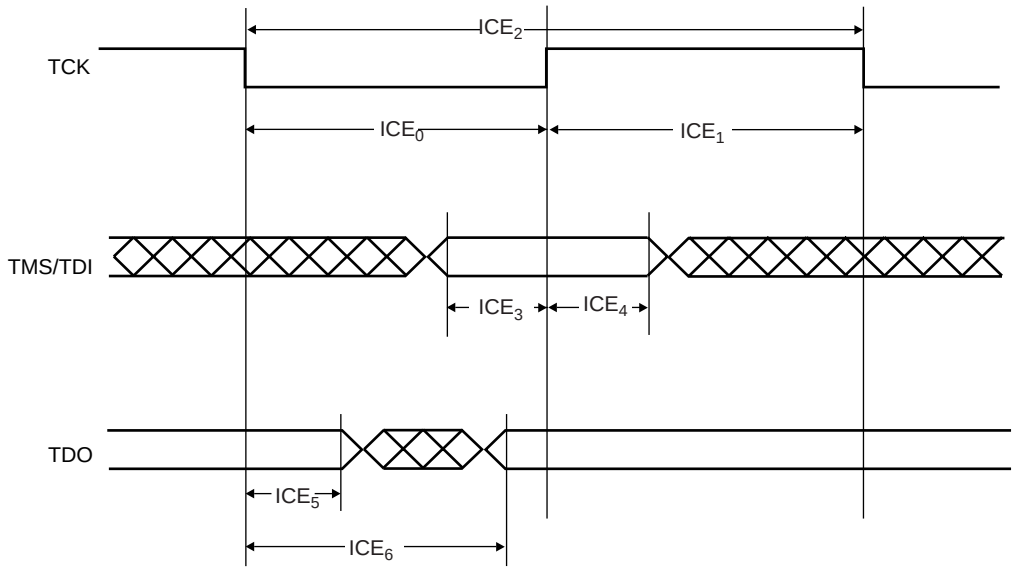
ICE Interface Signals

Table 85 shows timings relative to operating condition limits defined in the section “Conditions and Timings Computation” on page 439.

Table 85. ICE Interface Timing specification

Symbol	Parameter	Conditions	Min	Max	Units
ICE_0	TCK Low Half-period		24.0		ns
ICE_1	TCK High Half-period		24.0		ns
ICE_2	TCK Period		48.0		ns
ICE_3	TDI, TMS, Setup before TCK High		1.1		ns
ICE_4	TDI, TMS, Hold after TCK High		0		ns
ICE_5	TDO Hold Time	C_{TDO} = 0 pF	4.3		ns
		C_{TDO} derating	0.037		ns/pF
ICE_6	TCK Low to TDO Valid	C_{TDO} = 0 pF		10.7	ns
		C_{TDO} derating		0.037	ns/pF

Figure 181. ICE Interface Signals



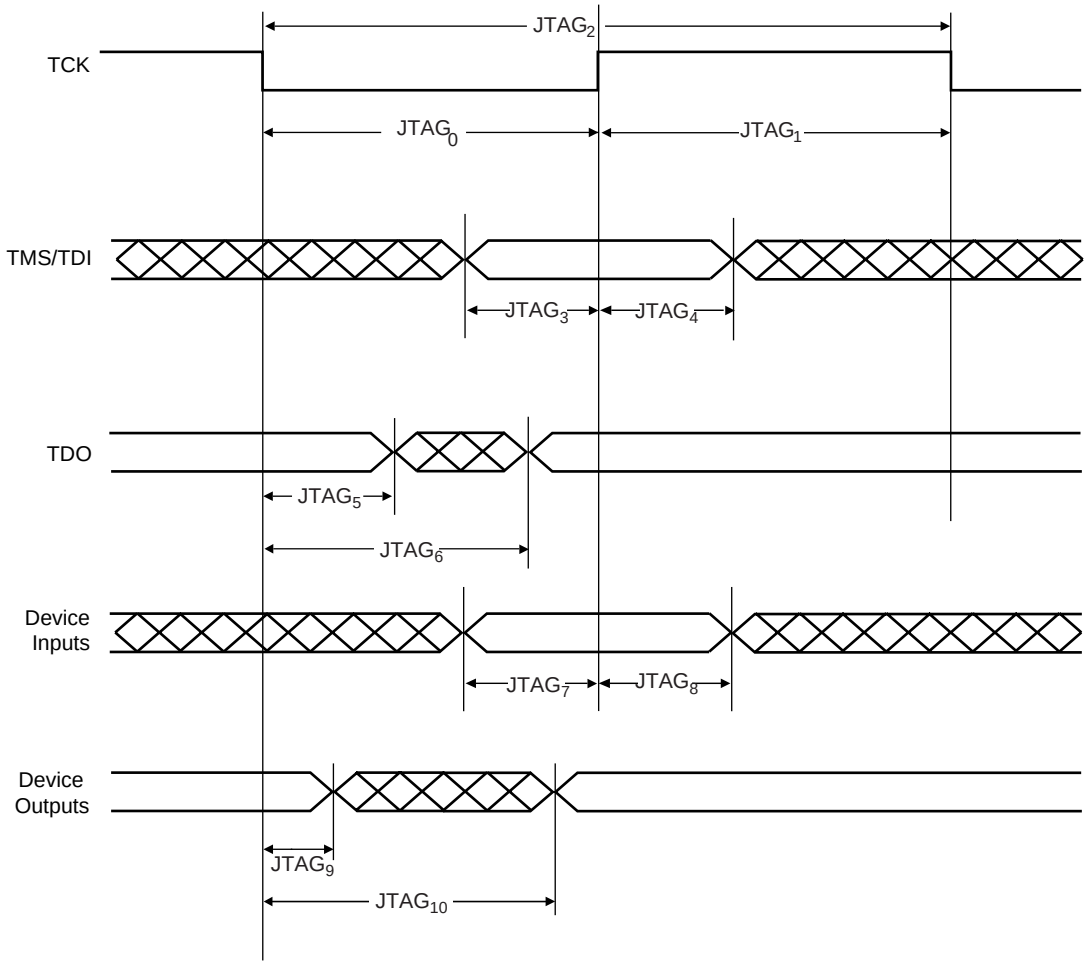
JTAG Interface Signals

Table 86 shows timings relative to operating condition limits defined in the section “Conditions and Timings Computation” on page 439.

Table 86. JTAG Interface Timing specification

Symbol	Parameter	Conditions	Min	Max	Units
$JTAG_0$	TCK Low Half-period		6.5		ns
$JTAG_1$	TCK High Half-period		5.5		ns
$JTAG_2$	TCK Period		12.0		ns
$JTAG_3$	TDI, TMS Setup before TCK High		0.6		ns
$JTAG_4$	TDI, TMS Hold after TCK High		1.5		ns
$JTAG_5$	TDO Hold Time	C_{TDO} = 0 pF	2.4		ns
		C_{TDO} derating	0.037		ns/pF
$JTAG_6$	TCK Low to TDO Valid	C_{TDO} = 0 pF		6.2	ns
		C_{TDO} derating		0.037	ns/pF
$JTAG_7$	Device Inputs Setup Time		0		ns
$JTAG_8$	Device Inputs Hold Time		3.0		ns
$JTAG_9$	Device Outputs Hold Time	C_{OUT} = 0 pF	2.7		ns
		C_{OUT} derating	0.035		ns/pF
$JTAG_{10}$	TCK to Device Outputs Valid	C_{OUT} = 0 pF		9.0	ns
		C_{OUT} derating		0.035	ns/pF

Figure 182. JTAG Interface Signals



Mechanical Characteristics

Thermal Data

In Table 87, 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 445.)

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 87. MTBF Versus Junction Temperature

Junction Temperature (T_J) (°C)	Estimated Lifetime (MTBF) (Year)
100	9
125	5
150	2
175	1

Table 88 summarizes the thermal resistance data depending on the package.

Table 88. Thermal Resistance Data

Symbol	Parameter	Condition	Package	Typ	Unit
θ_{JA}	Junction-to-ambient thermal resistance	Still Air	LQFP100	40.2	°C/W
θ_{JC}	Junction-to-case thermal resistance		LQFP100	13.1	

Junction Temperature

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

- $T_J = T_A + (P_D \times \theta_{JA})$

- $T_J = T_A + (P_D \times (\theta_{HEATSINK} + \theta_{JC}))$

where:

- θ_{JA} = package thermal resistance, Junction-to-ambient (°C/W), provided in Table 88 on page 445
- θ_{JC} = package thermal resistance, Junction-to-case thermal resistance (°C/W), provided in Table 88 on page 445
- $\theta_{HEAT\ SINK}$ = cooling device thermal resistance (°C/W), provided in the device datasheet
- P_D = device power consumption (W) estimated from data provided in the section “Power Consumption” on page 434
- 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 Drawing

Figure 183. 100-lead LQFP Package Drawing

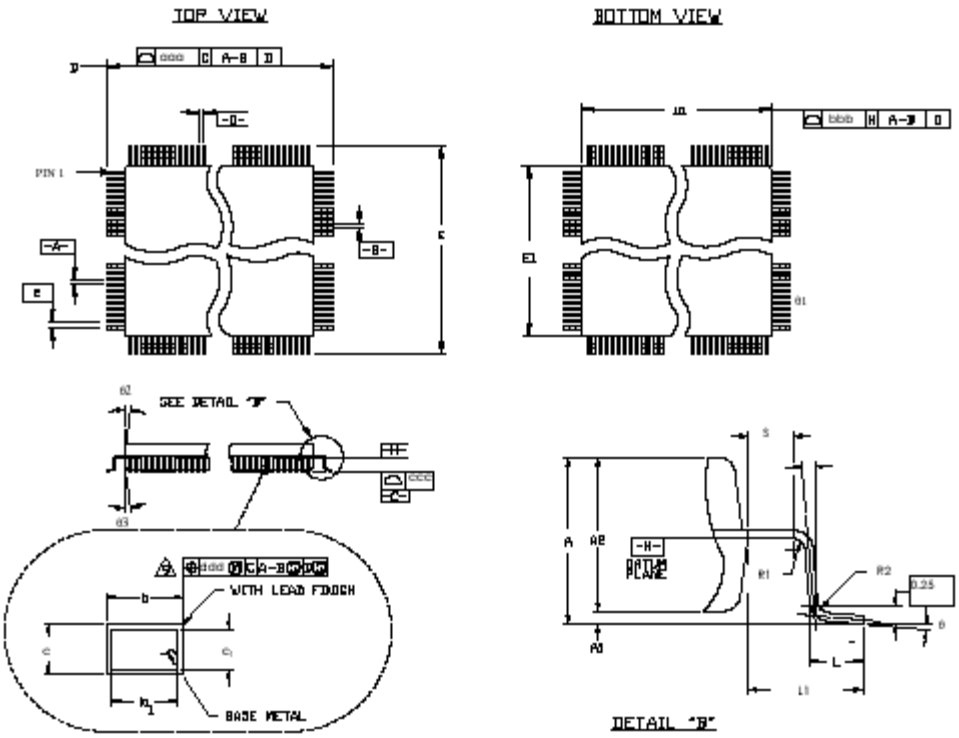


Table 89. 100-lead LQFP Package Dimensions (in mm)

Symbol	Min	Nom	Max	Symbol	Min	Nom	Max
c	0.09		0.2	B	0.17	0.22	0.27
c1	0.09		0.16	b1	0.17	0.2	0.23
L	0.45	0.6	0.75	**Tolerances of Form and Position**			
L1		1.00 REF		aaa		0.2	
R2	0.08		0.2	bbb		0.2	
R1	0.08			**BSC**			
S	0.2			D	16.0		
θ	0°	3.5°	7°	D1	14.0		
θ1	0°			E	16.0		
θ2	11°	12°	13°	E1	14.0		
θ3	11°	12°	13°	e	0.50		
A			1.6				
A1	0.05		0.15	ccc	0.10		
A2	1.35	1.4	1.45	ddd	0.06		

AT91RM3400 Ordering Information

Table 90. Ordering Information

Ordering Code	Package	ROM Code Revision	Temperature Operating Range
AT91RM3400-AI-001	LQFP 100	001	Industrial (-40°C to 85°C)

Table of Contents

Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131
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
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131
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
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
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

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

1790A–ATARM–11/03