Features

- DIOPSIS® Dual Core System Integrating an ARM926EJ-S™ ARM® Thumb® Processor Core and a MagicV of VLIW Magic DSP™ is optimized for Audio, Communication and Beam-forming Applications
- High Performance MagicV VLIW DSP
  - 1 GFLOPS - 1.6 Gops at 100 MHz
  - AHB Master Port, integrated DMA Engine and AHB Slave Port
  - Up to 10 Arithmetic Operations per Cycle (4 Multiply, 2 Add/Subtract, 1 Add, 1 Subtract 40-bit Floating Point and 32-bit Integer) allowing Single Cycle FFT Butterfly
  - Native Support for Complex Arithmetic and Vectorial SIMD Operations: One Complex Multiply with Dual Add/Sub per Clock Cycle or Two Multiply and Two Add/sub or Simple Scalar Operations
  - 32-bit Integer and IEEE® 40-bit Extended Precision Floating Point Numeric Format
  - 16-port Data Register File: 256 Registers organized in Two 128-register Banks
  - 5-issue predicated VLIW Architecture with Orthogonal ISA, Code Compression and Hardware Support for Code Efficient Software Pipeline Loops
  - 6 Accesses per Cycle Data Memory System (4 Accesses per Cycle for VLIW Operations + 2 Accesses per Cycle for DMA Transfers) supported by Flexible Addressing Capability
  - 2 Independent Address Generation Units Operating on a 64-register Address Register File Supporting Complex or Micro-Vectorial Accesses and DSP features: Programmable Stride and Circular Buffers
  - 1.7 Mbits of On-chip SRAM:
  - 16 K x 40-bit Data Memory Locations (6 Memory Accesses per Cycle)
  - 8 K x 128-bit Dual Port Program Memory Location, Equivalent to ~50K DSP Assembler Instructions (typical) thanks to Code Compression and SW Pipelining
  - DMA Access to the External Program and Data Memory
  - Three Main Operating Modes: Run, Debug and Sleep
  - User Mode and Privileged Interrupt Service Mode
  - Efficient Optimizing Assembler and C-Oriented Architecture: allows Easy Exploitation of the available Hardware Parallelism
  - ARM926EJ-S ARM Thumb Processor
  - DSP Instruction Extensions
  - ARM Jazelle® Technology for Java® Acceleration
  - 16-KByte Data Cache, 16-KByte Instruction Cache, Write Buffer
  - 220MIPS at 200MHz
  - Memory Management Unit
  - EmbeddedICE™ In-circuit Emulation, Debug Communication Channel Support
- Additional Embedded Memories
  - 32-KByte of internal ROM, two-cycle access at maximum bus speed
  - 48-KByte of internal SRAM, single-cycle access at maximum processor or bus speed
- External Bus Interface (EBI)
  - Supports SDRAM, Static Memory, SmartMedia™ and NAND Flash, CompactFlash™
- USB
  - USB 2.0 Full Speed (12 Mbits per second) Host Double Port
  - Dual On-chip Transceivers
  - Integrated FIFOs and Dedicated DMA Channels

DIOPSIS 940HF
ARM926EJ-S PLUS ONE GFLOPS DSP

AT572D940HF

Preliminary

7010A–DSP–07/08

- USB 2.0 Full Speed (12 Mbits per second) Device Port
  - On-chip Transceiver, 2-Kbyte Configurable Integrated FIFOs
  - Two dedicated PDC channels
- Ethernet MAC 10/100
  - Reduced Media Independent Interface (RMII) to Physical Layer
  - Integrated DMA channel
- AHB bus Matrix
  - Seven Masters and Five Slaves Handled
  - Boot Mode Select Option
  - Remap Command
- System Controller (SYSC)
  - Reset Controller
  - Periodic Interval Timer, Watchdog and Real-Time Timer
- Power Management Controller (PMC)
  - Very Slow Clock (32768Hz) Operating Mode
  - Software Programmable Power Optimization Capabilities
  - 3 to 20 MHz On-chip Oscillator and two PLLs
  - Four Programmable External Clock Signals
- Advanced Interrupt Controller (AIC)
  - Individually Maskable, Eight-level Priority, Vectored Interrupt Sources
  - Three External Interrupt Sources and One Fast Interrupt Source, Spurious Interrupt Protected
- Three 32-bit Parallel Input/Output Controllers (PIO)
  - 96 Programmable I/O Lines Multiplexed with up to Two Peripheral I/Os
  - Input Change Interrupt Capability on Each I/O Line
  - Individually Programmable Open-drain, Pull-up resistor and Synchronous Output
- Twenty-three Peripheral Data Controller (PDC) Channels
- Debug Unit (DBGU)
  - 2-wire USART and support for Debug Communication Channel, Programmable ICE Access Prevention
  - Two dedicated PDC channels
- Four Synchronous Serial Controllers (SSC)
  - Two Independent Clock and Frame Sync Pair 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
  - Two dedicated PDC channels for each SSC
- Three Universal Synchronous/Asynchronous Receiver Transmitters (USART)
  - Individual Baud Rate Generator, IrDA® Infrared Modulation/Demodulation
  - Support for ISO7816 T0/T1 Smart Card, Hardware and Software Handshaking, RS485 Support
  - Two dedicated PDC channels for each USART
- Two Master/Slave Serial Peripheral Interface (SPI)
  - 8- to 16-bit Programmable Data Length, Four External Peripheral Chip Selects
  - Two dedicated PDC Channels for each SPI
- One 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 Two-wire Interfaces (TWI)
  - Master Mode Support, All Atmel Two-wire EEPROMs Supported
- Two CAN Interfaces
  - Fully compliant with CAN 2.0 Part A and 2.0 Part B
- Multimedia Card Interface (MCI)
  - Automatic Protocol Control and Fast Automatic Data Transfers with PDMA, MMC and SDCard Compliant

- **IEEE 1149.1 JTAG Boundary Scan on All Digital Pins**
- **Required Power Supplies:**
  - **1.1V / 1.2V for VDDCORE and VDDOSC**
  - **3.3V for VDDPLLA**
  - **3.3V for VDDIOP (Peripheral I/Os) and for VDDIOM (Memory I/Os)**
- **Available in 324-ball CABGA Package**
- **Efficient ARM - DSP Interface through AHB master and slave ports, Memory Mapped Registers and Ports, Interrupt Lines and Semaphores**

1. Description

DIOPSIS 940HF is a Dual CPU Processor integrating a MagicV VLIW DSP and an ARM926EJ-S RISC MCU, plus a 370 Kbyte SRAM. The system combines the flexibility of the ARM926™ RISC controller with the very high performance of the DSP.

MagicV is a high performance VLIW DSP delivering 1 Giga floating-point operations per second (GFLOPS) and 1.6 Gops at 100 MHz clock rate. It is equipped with an AHB master port and an AHB slave port for system-on-chip integration. It has 256 data registers, 64 address registers, 10 independent arithmetic operating units, 2 independent address generation units and a DMA engine. To sustain the internal parallelism, the data bandwidth among the Register File, the Operators and the Data Memory System, is 80 bytes/cycle. The Data Memory System is designed to transfer 28 bytes/cycle. For instance, MagicV can produce a complete FFT butterfly per cycle by activating all the computing units; it operates on IEEE 754 40-bit extended precision floating-point and 32-bit integer numeric format for numerical computations, while internal memory accesses are supported by a powerful 16-bit MAGU (Multiple Address Generation Unit). It has also on-chip 16K x 40-bit 6-access/cycle data memory system and 8K x 128-bit dual port program memory locations. Efficient usage of the internal program memory is achieved through a general purpose code compression mechanism and a software pipelining support for systematic loops.

A C-oriented Architecture and an optimizing assembler facilitate the user in dealing with the parallelism of the processor resources and drastically simplify the code development. A rich library of C-callable DSP routines is available.

The ARM926 embedded micro controller core is a member of the Advanced RISC Machines (ARM) family of general purpose 32-bit microprocessors, which offer high performance and very low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles; the instruction set and the related decode mechanism are much simpler than the micro programmed Complex Instruction Set Computers.

The result of this simplicity is a high instruction throughput and an impressive real-time interrupt response. The ARM926 supports 16-bit Thumb subset of the most commonly used 32-bit instructions. These are expanded at run time with no degradation of the system performance. This gives 16-bit code density (saving memory area and cost) coupled with a 32-bit processor performance.

A rich set of peripherals and a 48 Kbyte internal memory provide a highly flexible and integrated system solution.

The ARM926EJ-S supports Jazelle Technology for Java acceleration.

2. Ball Configuration

Table 2-1. AT572D940HF Ball Assignment (I/O: 191 balls)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
A0/NBS0	B2	D5	K7	NCS2	B7	PIOA27	G9
A1/NBS2/NWR2	C2	D6	K5	NCS3/SM_NCS	E7	PIOA28	J9
A2	C1	D7	K1	NRD/NOE/CF_NOE	B6	PIOA29	A8
A3	D4	D8	K2	NRST	J17	PIOA30	D8
A4	D3	D9	K6	NWR0/NWE/CF_NWE	C6	PIOA31	B8
A5	D1	D10	K8	NWR1/NBS1/CF_NIOR	D6	PIOB0	U8
A6	E4	D11	L5	NWR3/NBS3/CF_NIOW	G7	PIOB1	L9
A7	E3	D12	L1	PIOA0	F11	PIOB2	P9
A8	F6	D13	L2	PIOA1	C11	PIOB3	R9
A9	G6	D14	L4	PIOA2	A11	PIOB4	V9
A10	F3	D15	L7	PIOA3	B11	PIOB5	L10
A11	H8	D16	M3	PIOA4	H10	PIOB6	N10
A12	F2	D17	L8	PIOA5	G10	PIOB7	V10
A13	F1	D18	M4	PIOA6	D10	PIOB8	T10
A14	G3	D19	M5	PIOA7	B17	PIOB9	P10
A15	H7	D20	M6	PIOA8	A17	PIOB10	M10
A16/SD_BA0	G1	D21	N1	PIOA9	B16	PIOB11	N11
A17/SD_BA1	G2	D22	M7	PIOA10	A16	PIOB12	M11
A18	H6	D23	N4	PIOA11	C15	PIOB13	L11
A19	H3	D24	N5	PIOA12	H17	PIOB14	U12
A20	J8	D25	P1	PIOA13	V15	PIOB15	T12
A21	H2	D26	P3	PIOA14	U15	PIOB16	R12
A_JCFG	N16	D27	P4	PIOA15	V16	PIOB17	N12
A_RTCK	M17	D28	P5	PIOA16	T15	PIOB18	V13
A_TCK	N17	D29	R1	PIOA17	V17	PIOB19	U13
A_TDI	M14	D30	R2	PIOA18	T16	PIOB20	T13
A_TDO	M16	D31	R3	PIOA19	T17	PIOB21	P13
A_TMS	N15	M_NTRST	E16	PIOA20	U18	PIOB22	V14
A_NTRST	M13	M_TCK	F13	PIOA21	T18	PIOB23	R14
D0	H1	M_TDI	E15	PIOA22	R15	PIOB24	J10
D1	J7	M_TDO	E14	PIOA23	R18	PIOB25	H15
D2	J2	M_TMS	E17	PIOA24	H16	PIOB26	B12
D3	J1	NCS0	F7	PIOA25	B9	PIOB27	A12
D4	K9	NCS1/SD_CS	A6	PIOA26	D9	PIOB28	F9

Table 2-1. AT572D940HF Ball Assignment (I/O: 191 balls) (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
PIOB29	B10	PIOC11	L13	PIOC25	K15	SD_NWE	B4
PIOB30	A10	PIOC12	L18	PIOC26	K11	TEST	J18
PIOB31	A9	PIOC13	K12	PIOC27	K10	USBD_M	N8
PIOC0	D15	PIOC14	H13	PIOC28	E12	USBD_P	P8
PIOC1	D14	PIOC15	G17	PIOC29	D12	USBHA_M	R7
PIOC2	C14	PIOC16	G18	PIOC30	P16	USBHA_P	T7
PIOC3	D13	PIOC17	G14	PIOC31	P17	USBHB_M	U7
PIOC4	C13	PIOC18	F17	PLL_RCA	U2	USBHB_P	V7
PIOC5	G12	PIOC19	H14	PLL_RCB	P6	XIN	U5
PIOC6	F12	PIOC20	F16	SD_A10	A7	XOUT	V5
PIOC7	G13	PIOC21	E18	SD_CK	B5	X32EN	N7
PIOC8	F18	PIOC22	K14	SD_CKE	C5	X32IN	V2
PIOC9	M18	PIOC23	K16	SD_NCAS	A4	X32OUT	V3
PIOC10	L12	PIOC24	K17	SD_NRAS	D5		

Table 2-2. AT572D940HF Ball Assignment (Power and Ground: 128 balls)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
VDDCORE	F4	VDDIOM	B3	VDDIOP	T9	VDDPLLA	T3
VDDCORE	J4	VDDIOM	E5	VDDIOP	V8	GND	D2
VDDCORE	L6	VDDIOM	E1	VDDIOP	F14	GND	E2
VDDCORE	T2	VDDIOM	G4	VDDIOP	G16	GND	F5
VDDCORE	M9	VDDIOM	H4	VDDIOP	H18	GND	G5
VDDCORE	P11	VDDIOM	J5	VDDIOP	J15	GND	H5
VDDCORE	T14	VDDIOM	K3	VDDIOP	K13	GND	J6
VDDCORE	N13	VDDIOM	M2	VDDIOP	L16	GND	J3
VDDCORE	L15	VDDIOM	N3	VDDIOP	M12	GND	K4
VDDCORE	J13	VDDIOM	P2	VDDIOP	N14	GND	L3
VDDCORE	H11	VDDIOMP	E9	VDDIOP	U17	GND	M1
VDDCORE	D16	VDDIOMP	G8	VDDIOP	P14	GND	N2
VDDCORE	E13	VDDIOP	C10	VDDIOP	P12	GND	N6
VDDCORE	H9	VDDIOP	D11	VDDIOP	U11	GND	R4
VDDCORE	E8	VDDIOP	G11	VDDIOP	R10	GND	T1
VDDCORE	A2	VDDIOP	A13	VDDIOP	V6	GND	T8
VDDIOM	D7	VDDIOP	A15	VDDOSC32	U4	GND	R8
VDDIOM	A5	VDDIOP	C16	VDDOSCM	R5	GND	N9

Table 2-2. AT572D940HF Ball Assignment (Power and Ground: 128 balls) (Continued)

Name	Pin	Name	Pin	Name	Pin	Name	Pin
GND	U10	GND	J14	GND	E10	GND	R16
GND	V11	GND	J12	GND	C9	GND	R17
GND	R11	GND	H12	GND	C8	GND	M15
GND	V12	GND	G15	GND	C7	GND	C18
GND	R13	GND	F15	GND	E6	GND	C17
GND	U14	GND	D18	GND	A3	GND	B18
GND	U16	GND	D17	GND	C4	GND	C3
GND	P15	GND	B15	GND	U6	GND	B1
GND	P18	GND	B14	GND	V4	GND	T6
GND	N18	GND	B13	GND	M8	GND	R6
GND	L14	GND	C12	GND	U9	GND	J11
GND	J16	GND	E11	GND	T11	GNDOSC32	T5
GND	L17	GND	F8	GND	U1	GNDOSCM	P7
GND	K18	GND	F10	GND	A14	GNDPLLA	U3

All pins not listed in Table 2-1 and Table 2-2 are “not connected”.

2.1 Pin Name Conventions

Pin names are built using the following structure:

(functional block name) _ (activity level) (line name) (bus index)

where:

functional block name = name of the related functional block (when not a global function)

activity level = “N” for low active lines; blank for high active lines

line name = name of the pin line function

bus index = number corresponding to the index when the pin line is a bus element

3. Pin Description

Table 3-1. AT572D940HF Pin Description

Module	Name	Function	Type	Active Level	Notes
AIC	EXT_IRQ0 - EXT_IRQ2	External Interrupt Request	bi-03		input through PIO line
AIC	M_MODE	Interrupt Request from MagicV	bi-03		output through PIO line
AIC	M_SIRQ0 - M_SIRQ3	Generic Interrupt Request from MagicV	bi-03		output through PIO line
A JTAG	A_JCFG	ARM JTAG / Chip Boundary Scan select	in		internal pull-down resistor (low = ARM JTAG selected)
A JTAG	A_RTCK	ARM JTAG Returned Test Clock	out-03		
A JTAG	A_TCK	ARM JTAG Test Clock	in		no pull-up resistor
A JTAG	A_TDI	ARM JTAG Test Data Input	in		no pull-up resistor
A JTAG	A_TDO	ARM JTAG Test Data Output	out-03		
A JTAG	A_TMS	ARM JTAG Test Mode Select	in		no pull-up resistor
CAN	CAN0_RX	CAN 0 bus Data in	bi-03		input through PIO line
CAN	CAN0_TX	CAN 0 bus Data out	bi-03		output through PIO line
CAN	CAN1_RX	CAN 1 bus Data in	bi-03		input through PIO line
CAN	CAN1_TX	CAN 1 bus Data out	bi-03		output through PIO line
CF Logic	CF_NCE1- CF_NCE2	CompactFlash Chip Enable	bi-03	low	output through PIO line
CF Logic	CF_NOE	CompactFlash Output Enable	out-03	low	
CF Logic	CF_NWE	CompactFlash Write Enable	out-03	low	
CF Logic	CF_NIOR	CompactFlash IO Read	out-03	low	
CF Logic	CF_NIOW	CompactFlash IO Write	out-03	low	
CF Logic	CF_RNW	CompactFlash Read Not Write	bi-03		output through PIO line
CF Logic	CF_NCS0 - CF_NCS1	CompactFlash Chip Select	bi-03	low	output through PIO line
DBGU	DBG_RXD	Debug Serial Line Data in	bi-03		input through PIO line
DBGU	DBG_TXD	Debug Serial Line Data out	bi-03		output through PIO line
EBI	A0 - A21	Address Bus	out-03		0 at reset
EBI	A22 - A25	Address Bus	bi-03		output through PIO line 0 at reset
EBI	D0- D31	Data Bus	bi-03		pulled-up input at reset
EBI	NWAIT	External Wait Signal		low	input through PIO line
EBI	BMS	Boot Memory Select	bi-03		input through PIO line 1→ external boot selected 0→ internal boot selected
ETH	E_RXER	Ethernet RMII Receive Error	bi-03		input through PIO line

Table 3-1. AT572D940HF Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
ETH	E_TXD0 - E_TXD1	Ethernet RMII Transmit Data Bus	bi-03		output through PIO line
ETH	E_TXEN	Ethernet RMII Transmit Enable	bi-03		output through PIO line
ETH	E_REFCK	Ethernet RMII Reference Clock	bi-03		input through PIO line
ETH	E_CRSDV	Ethernet RMII Carrier Sense/Data Valid	bi-03		input through PIO line
ETH	E_RXD0 - E_RXD1	Ethernet RMII Receive Data Bus	bi-03		input through PIO line
ETH	E_FCE100	Ethernet RMII Force 100 Mb/s operation	bi-03	high	output through PIO line
ETH	E_MDIO	Ethernet RMII PHY Management Data	bi-03		through PIO line
ETH	E_MDCK	Ethernet RMII PHY Management Clock	bi-03		output through PIO line
MCI	MCCK	Multimedia Card Clock	bi-03		through PIO line
MCI	MCCDA	Multimedia Card Command	bi-03		through PIO line
MCI	MCDA0-MCDA3	Multimedia Card Data	bi-03		through PIO line
M JTAG	M_NTRST	MagicV JTAG Test Reset	in		
M JTAG	M_TCK	MagicV JTAG Test Clock	in		no pull-up resistor
M JTAG	M_TDI	MagicV JTAG Test Data Input	in		no pull-up resistor
M JTAG	M_TDO	MagicV JTAG Test Data Output	out-03		
M JTAG	M_TMS	MagicV JTAG Test Mode Select	in		no pull-up resistor
OSC	XIN	Main Oscillator Quartz	in		
OSC	XOUT	Main Oscillator Quartz	out		
OSC	X32IN	Slow Clock Oscillator Quartz	in		
OSC	X32OUT	Slow Clock Oscillator Quartz	out		
OSC	X32EN	Slow Clock Oscillator Enable	in	high	internal pull-up resistor (internal oscillator enabled)
PIO A	PIOA0 - PIOA31	Parallel Input/Output A	bi-03		general purpose programmable I/Os or peripheral I/Os; Pulled-up input at reset
PIO B	PIOB0 - PIOB31	Parallel Input/Output B	bi-03		general purpose programmable I/Os or peripheral I/Os; Pulled-up input at reset
PIO C	PIOC0 - PIOC31	Parallel Input/Output C	bi-03		general purpose programmable I/Os or peripheral I/Os; Pulled-up input at reset
PLL	PLL_RCA	PLL A Filter	in		
PLL	PLL_RCB	PLL B Filter	in		to be left floating (test input)
PMC	A_CK	ARM Clock	bi-03		output through PIO line for test purpose

Table 3-1. AT572D940HF Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
PMC	M_CK	MagicV Clock	bi-03		output through PIO line for test purpose
PMC	P_CK0-P_CK3	Programmable Clock	bi-03		output through PIO line
SDRAMC	SDCK	SDRAM Clock Output	out-03		
SDRAMC	SD_CKE	SDRAM Clock Enable	out-04	high	
SDRAMC	SD_NCS	SDRAM Chip Select	out-03	low	
SDRAMC	SD_BA0 - SD_BA1	SDRAM Bank Select	out-03		
SDRAMC	SD_NWE	SDRAM Write Enable	out-04	low	
SDRAMC	SD_NRAS - SD_NCAS	Row and Column Address Strobe	out-04	low	
SDRAMC	SD_A10	SDRAM Bus Address bit 10	out-04		
SMC	NCS0 - NCS3	Chip Select Signal	out-03	low	1 at reset;
SMC	NCS4 - NCS7	Chip Select Signal	bi-03	low	1 at reset output through PIO line
SMC	NWR0 - NWR3	Write Signal	out-03	low	1 at reset
SMC	NOE	Output Enable	out-03	low	1 at reset
SMC	NRD	Read Signal	out-03	low	1 at reset
SMC	NWE	Write Enable	out-03	low	1 at reset
SMC	NBS0 - NBS3	Byte Select	out-03	low	1 at reset
SM Logic	SM_NOE	SmartMedia Output Enable	bi-03	low	output through PIO line
SM Logic	SM_NWE	SmartMedia Write Enable	bi-03	low	output through PIO line
SPI	SPI0_MOSI	SPI 0 Master Out/Slave In data	bi-03		through PIO line SPI SLV → data input SPI MST → data output
SPI	SPI0_MISO	SPI 0 Master In/Slave Out data	bi-03		through PIO line SPI SLV → data output SPI MST → data input
SPI	SPI0_NCS0	SPI 0 Input/Output Chip select	out-03	low	through PIO line SPI SLV → CS Input SPI MST → CS 0 Output
SPI	SPI0_NCS1 - SPI0_NCS3	SPI 0 Output Chip Selects	bi-03	low	output through PIO line SPI SLV → n.a. SPI MST → CS 3, 2, 1 Outputs
SPI	SPI0_CK	SPI 0 Serial clock	bi-03		through PIO line SPI SLV → clock input SPI MST → clock output

Table 3-1. AT572D940HF Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
SPI	SPI1_MOSI	SPI 1 Master Out/Slave In data	bi-03		through PIO line SPI SLV → data input SPI MST → data output
SPI	SPI1_MISO	SPI 1 Master In/Slave Out data	bi-03		through PIO line SPI SLV → data output SPI MST → data input
SPI	SPI1_NCS0	SPI 1 Input/Output Chip select	out-03	low	through PIO line SPI SLV → CS Input SPI MST → CS 0 Output
SPI	SPI1_NCS1 - SPI1_NCS3	SPI 1 Output Chip Selects	bi-03	low	output through PIO line SPI SLV → n.a. SPI MST → CS 3, 2, 1 Outputs
SPI	SPI1_CK	SPI 1 Serial clock	bi-03		through PIO line SPI SLV → clock input SPI MST → clock output
SSC	SSC0_TXD	Synchronous Serial Controller 0 Data Out	bi-03		output through PIO line
SSC	SSC0_RXD	Synchronous Serial Controller 0 Data In	bi-03		input through PIO line
SSC	SSC0_TF	Synchronous Serial Controller 0 Transmit Frame Clock	bi-03		through PIO line
SSC	SSC0_RF	Synchronous Serial Controller 0 Receive Frame Clock	bi-03		through PIO line
SSC	SSC0_TK	Synchronous Serial Controller 0 Transmit Bit Clock	bi-03		through PIO line
SSC	SSC0_RK	Synchronous Serial Controller 0 Receive Bit Clock	bi-03		through PIO line
SSC	SSC1_TXD	Synchronous Serial Controller 1 Data Out	bi-03		output through PIO line
SSC	SSC1_RXD	Synchronous Serial Controller 1 Data In	bi-03		input through PIO line
SSC	SSC1_TF	Synchronous Serial Controller 1 Transmit Frame Clock	bi-03		through PIO line
SSC	SSC1_RF	Synchronous Serial Controller 1 Receive Frame Clock	bi-03		through PIO line
SSC	SSC1_TK	Synchronous Serial Controller 1 Transmit Bit Clock	bi-03		through PIO line
SSC	SSC1_RK	Synchronous Serial Controller 1 Receive Bit Clock	bi-03		through PIO line
SSC	SSC2_TXD	Synchronous Serial Controller 2 Data Out	bi-03		output through PIO line
SSC	SSC2_TF	Synchronous Serial Controller 2 Transmit Frame Clock	bi-03		through PIO line

Table 3-1. AT572D940HF Pin Description (Continued)

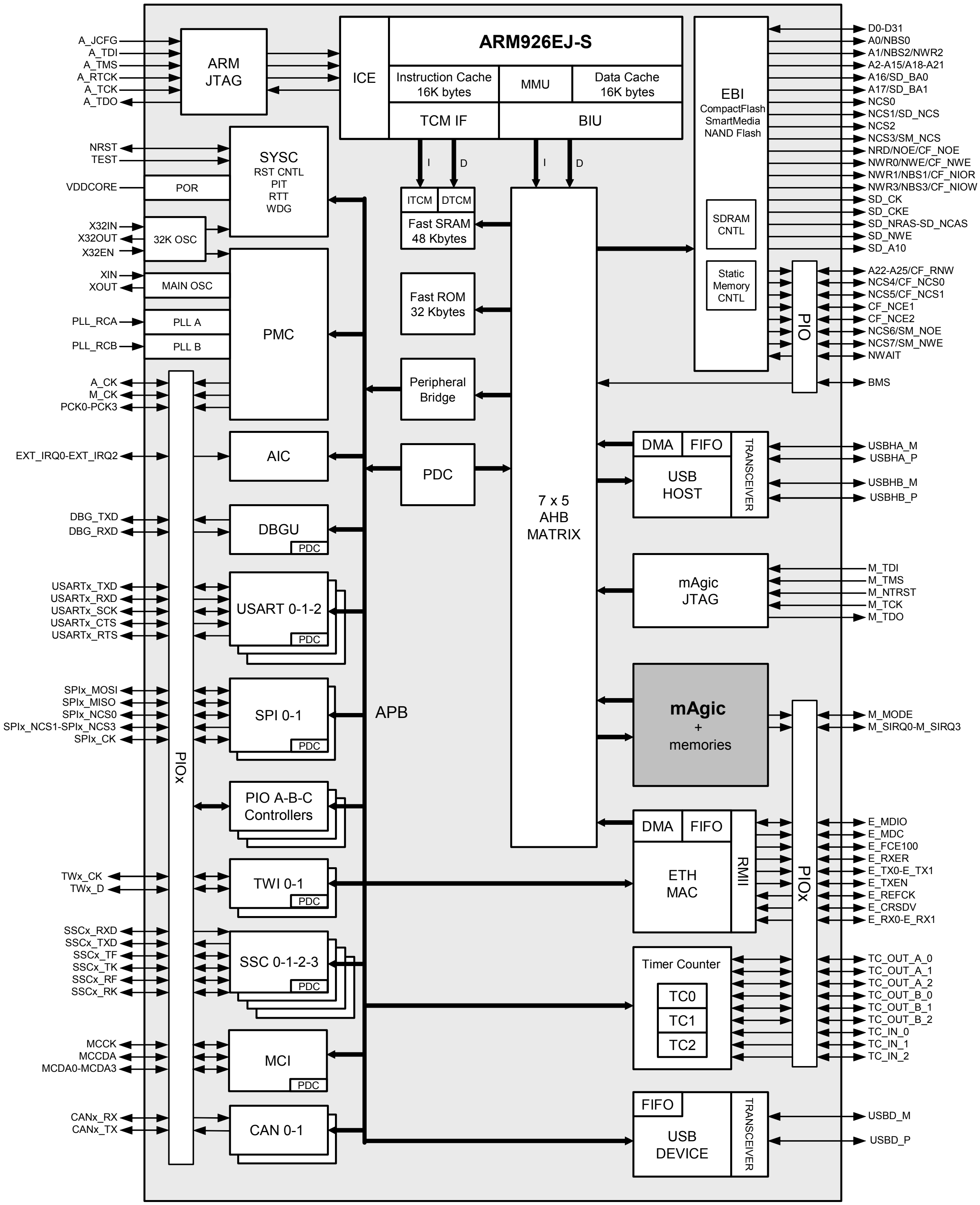
Module	Name	Function	Type	Active Level	Notes
SSC	SSC2_RF	Synchronous Serial Controller 2 Receive Frame Clock	bi-03		through PIO line
SSC	SSC2_TK	Synchronous Serial Controller 2 Transmit Bit Clock	bi-03		through PIO line
SSC	SSC2_RK	Synchronous Serial Controller 2 Receive Bit Clock	bi-03		through PIO line
SSC	SSC2_RXD	Synchronous Serial Controller 2 Data In	bi-03		input through PIO line
SSC	SSC3_TXD	Synchronous Serial Controller 3 Data Out	bi-03		output through PIO line
SSC	SSC3_RXD	Synchronous Serial Controller 3 Data In	bi-03		input through PIO line
SSC	SSC3_TF	Synchronous Serial Controller 3 Transmit Frame Clock	bi-03		through PIO line
SSC	SSC3_RF	Synchronous Serial Controller 3 Receive Frame Clock	bi-03		through PIO line
SSC	SSC3_TK	Synchronous Serial Controller 3 Transmit Bit Clock	bi-03		through PIO line
SSC	SSC3_RK	Synchronous Serial Controller 3 Receive Bit Clock	bi-03		through PIO line
SYSC	NRST	Chip Reset	bi-03	low	open drain
TC	TC_OUT_A0	Timer Counter A out 0	bi-03		through PIO line
TC	TC_OUT_A1	Timer Counter A out 1	bi-03		bidirectional through PIO line
TC	TC_OUT_A2	Timer Counter A out 2	bi-03		bidirectional through PIO line
TC	TC_OUT_B0	Timer Counter B out 0	bi-03		bidirectional through PIO line
TC	TC_OUT_B1	Timer Counter B out 1	bi-03		bidirectional through PIO line
TC	TC_OUT_B2	Timer Counter B out 2	bi-03		bidirectional through PIO line
TC	TC_IN_0	Timer Counter in 0	bi-03		input through PIO line
TC	TC_IN_1	Timer Counter in 1	bi-03		input through PIO line
TC	TC_IN_2	Timer Counter in 2	bi-03		input through PIO line
TST	TEST	Test Mode Select	in	high	pull-down resistor (Functional Mode selected)
TWI	TW0_D	Two Wire 0 Data	bi-03		bidirectional through PIO line
TWI	TW0_CK	Two Wire 0 Clock	bi-03		bidirectional through PIO line
TWI	TW1_D	Two Wire 1 Data	bi-03		bidirectional through PIO line
TWI	TW1_CK	Two Wire 1 Clock	bi-03		bidirectional through PIO line
USBD	USBD_M	USB Device Port Data -	usb-bi		
USBD	USBD_P	USB Device Port Data +	usb-bi		
USBH	USBHA_M	USB Host Port A Data -	usb-bi		external 15K pull-down required

Table 3-1. AT572D940HF Pin Description (Continued)

Module	Name	Function	Type	Active Level	Notes
USBH	USBHA_P	USB Host Port A Data +	usb-bi		external 15K pull-down required
USBH	USBHB_M	USB Host Port B Data -	usb-bi		external 15K pull-down required
USBH	USBHB_P	USB Host Port B Data +	usb-bi		external 15K pull-down required
USART	USART0_RXD	USART 0 Data in	bi-03		input through PIO line
USART	USART0_TXD	USART 0 Data out	bi-03		bidirectional through PIO line
USART	USART0_SCK	USART 0 Serial clock	bi-03		bidirectional through PIO line for synchronous mode only
USART	USART0_CTS	USART 0 Clear to send	bi-03		input through PIO line
USART	USART0_RTS	USART 0 Request to send	bi-03		output through PIO line
USART	USART1_RXD	USART 1 Data in	bi-03		input through PIO line
USART	USART1_TXD	USART 1 Data out	bi-03		bidirectional through PIO line
USART	USART1_SCK	USART 1 Serial clock	bi-03		bidirectional through PIO line for synchronous mode only
USART	USART1_CTS	USART 1 Clear to send	bi-03		input through PIO line
USART	USART1_RTS	USART 1 Request to send	bi-03		output through PIO line
USART	USART2_RXD	USART 2 Data in	bi-03		input through PIO line
USART	USART2_TXD	USART 2 Data out	bi-03		bidirectional through PIO line
USART	USART2_SCK	USART 2 Serial clock	bi-03		bidirectional through PIO line for synchronous mode only
USART	USART2_CTS	USART 2 Clear to send	bi-03		input through PIO line
USART	USART2_RTS	USART 2 Request to send	bi-03		output through PIO line
Power	VDDCORE	Core power supply	Power		1.1V / 1.2V (nominal)
Power	VDDIOP	Peripherals I/O Lines Power Supply	Power		3.3V (nominal)
Power	VDDIOM	EBI I/O Lines Power Supply	Power		3.3V (nominal)
Power	VDDIOMP	EBI/Peripherals I/O Lines Power Supply	Power		3.3V (nominal)
Power	VDDOSC32	32KHz Oscillator Power Supply	Power		1.1V / 1.2V (nominal)
Power	VDDOSCM	Main Oscillator + PLLB Power Supply	Power		1.1V / 1.2V (nominal)
Power	VDDPLLA	PLLA power supply	Power		3.3V (nominal)
Ground	GND	Core and IO Ground	Ground		
Ground	GNDOSC32	32KHz Oscillator Ground	Ground		
Ground	GNDOSCM	Main Oscillator PLLB Ground	Ground		
Ground	GNDPLLA	PLLA Ground	Ground		

4. Block Diagram

Figure 4-1. AT572D940HF Architecture



5. Architectural Overview

DIOPSIS 940 HF (also named D940HF) is a dual-core processing platform for prosumer audio, speech processing, automotive sound and robotics applications, integrating a floating-point Magic DSP™ and an ARM926EJ-S RISC microprocessor.

The system combines the flexibility of the ARM926 RISC controller with the processing power of the mAgic VLIW floating-point DSP. This combination makes DIOPSIS suited for applications needing both control and intensive numerical applications. The 40-bit floating-point provides high dynamic range and maximum numerical precision, reducing time to market. DIOPSIS horse-power is fully exploited on complex domain applications, like frequency domain signal processing.

5.1 System management

The availability of a standard RISC on-chip reduces software development effort for the non critical and control segments of the application. ARM926 features an MMU for virtual memory and sophisticated memory protection, making it an ideal platform for operating systems such as Windows CE® or Linux®. This leaves MagicV fully available for the numerically intensive part of the application. The synchronization between the two processors can be either based on interrupts or on software polling on semaphores.

The ARM926 is the master processor of D940HF. The bootstrap sequence of the D940HF starts at the bootstrap of the ARM926 from its internal ROM or external non-volatile memory. The ARM then boots MagicV from a non-volatile memory. After bootstrap the D940HF can start its normal operations. The DSP side of many applications can be implemented on the D940HF by using only the internal memory. In fact, its 8K by 128-bit program memory coupled with the availability of the general purpose code compression and the software pipelining of systematic loops, gives an equivalent on-chip program memory size of about 24K cycles, corresponding to ~50K DSP assembler instructions (typically).

5.2 AMBA™ Architecture

The architecture is based on an AMBA bus: the multilayer AHB matrix and the APB.

The AHB matrix consists of seven masters:

0. ARM926 Instruction
1. ARM926-Data
2. Peripheral Data Controller (PDC)
3. MagicV
4. USB Host
5. Ethernet MAC 10/100
6. MagicV JTAG

and of five slaves:

0. ARM926 SRAM
1. ARM926 ROM
2. MagicV Registers and Memories + USB Host Registers
3. The External Bus Interface
4. The AHB-APB bridge

The following table defines the possible AHB MST-SLV links:

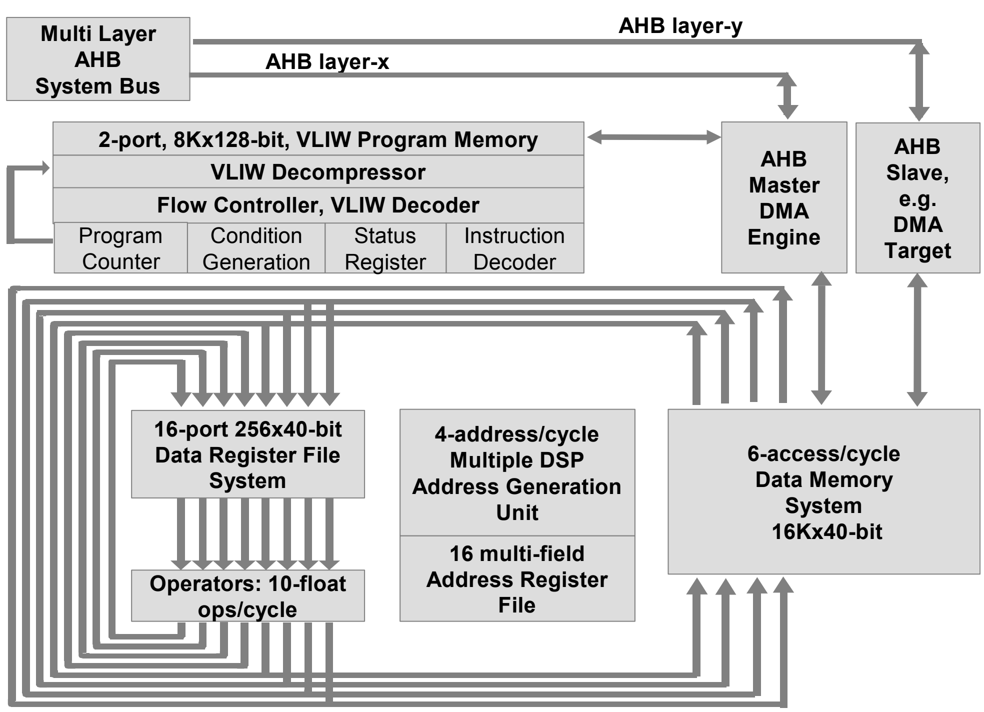
Table 5-1. AHB Masters-Slaves possible links

	MASTERS						
SLAVES	ARM-I	ARM-D	HPDC	MagicV DMA	USB HOST	ETH MAC	M-JTAG
ARM RAM	0 (default MST)	1	2	3	4	5	
ARM ROM	0 (default MST)	1					
MagicV-USBH		0 (default MST)	1				2
HEBI	0 (default MST)	1	2	3	4	5	6
HBRIDGE		0 (default MST)	1	2			

5.3 MagicV VLIW DSP Processor

The MagicV VLIW DSP is the numeric processor of D940HF. It operates on IEEE 754 40-bit extended precision floating-point and 32-bit integer numeric format. The main components of the DSP subsystem are the core processor, the on-chip memories, the DMA engine and its AHB master and slave interfaces. The operators block, the register file, the multiple address generation unit and the program decoding and sequencing unit are the computing part of the core processor. A short description of each block is given in the following paragraphs.

Figure 5-1. MagicV DSP Block Diagram



5.3.1 RISC-like VLIW DSP

MagicV is a Very Long Instruction Word engine, but from a user's point of view, it works like a RISC machine by implementing triadic computing operations on data coming from the register file and data move operations between the local memories and the register file. The operators are pipelined for maximum performance. The pipeline depth depends on the operator used. The scheduling and parallelism operations are automatically defined and managed at compile time by the assembler-optimizer, allowing efficient code execution. The architecture is designed for efficient C-language support.

5.3.2 Memories and Data Register File

MagicV Data Memory System contains 16K*40-bit on-chip memory locations supporting up to 6 accesses/cycle. 4-accesses/cycle are reserved for the activities driven by MagicV Multiple Address Generation unit: these accesses are reserved for the computing part of the core. An access/cycle is assigned to serve the DMA activity launched by the core itself through MagicV AHB master port. An additional access/cycle can be simultaneously requested by external devices through MagicV AHB slave port (e.g for data exchange with the interfaces of the ADC and the DAC converters).

The Program Memory stores the VLIW program to be executed by MagicV. It is an 8K-word by 128-bit dual port memory. A port is driven by the Flow Controller to fetch the compressed VLIW word. The other port is accessed by the DMA engine, supported by the AHB master interface, or by the external devices through MagicV AHB slave port.

To provide optimal data bandwidth and to give the best support to the RISC-like programming model, MagicV arithmetic computations are supported by a 16-ported, 256x40-bit entries, Data Register File System.

5.3.3 DSP Operators Block

The Operators Block contains the hardware that performs arithmetical operations. It works on 32-bit signed integers and IEEE 754 extended precision 40-bit floating-point data. The Operators Block is composed of four integer/floating point multipliers, an adder, a subtractor and two add-subtract integer/floating point units; moreover, it has two shift/logic units, a Min/Max operator and two seed generators for efficient division and inverse square root computation.

5.3.4 MagicV AHB master interface

MagicV VLIW DSP is equipped with an AHB master which supports MagicV DMA engine.

5.3.5 MagicV AHB slave interface

External AHB masters, like ARM and JTAG can access the memories and the registers of MagicV DSP through MagicV AHB slave interface. In Debug mode all the internal resources are memory mapped, while in run mode or sleep mode access restrictions apply. At every cycle, one port of the Data Memory System is reserved to read/store accesses performed through the AHB slave interface. Example of usage: data sampled by AD Converters can be written inside the MagicV Data Memory in parallel to the DMA (through the master port) and the VLIW operations.

5.3.6 ARM<->MagicV Interrupts

MagicV and the ARM can exchange synchronization signals based on interrupts to allow a tight coupling between their operations at run time.

5.4 ARM926 Processor

The ARM926 is a member of the ARM9™ family of general purpose microprocessors. The ARM926 is targeted at multi-tasking applications where full memory management, high performance and low power are important.

The ARM926 supports the 32-bit ARM and 16-bit THUMB instruction sets, enabling the user to trade off between high performance and high code density. The ARM926 includes features for efficient execution of Java byte codes.

The ARM926 supports the ARM debug architecture and includes logic to assist both the hardware and the software debug.

The ARM926 provides an integer core that supports the DSP instruction set extension.

The ARM926 supports virtual memory addressing through its standard ARM v4 and v5 memory management unit (MMU).

The ARM926 provides two independent AHB master interfaces for data and instruction.

The ARM926 provides two independent Tightly Coupled Memory (TCM) interfaces.

The ARM926 implements ARM architecture version 5TEJ with 5 stage pipeline.

The ARM926 embeds 16-Kbyte Data Cache and 16-Kbyte Instruction Cache.

5.4.1 ARM Memories

The ARM926 memories consist of:

- 32Kbyte ROM selectable as boot memory
- 48Kbyte Fast SRAM
  - Single Cycle Access at full bus speed
  - Supports ARM926EJ-S TCM interface at full processor speed
  - D-TCM and I-TCM programmable size

5.4.2 ARM Boot

The system always boots at address 0x0. The memory layout can be configured with two parameters to ensure a maximum number of possibilities for booting.

REMAP allows the user to lay out the first internal SRAM bank to 0x0 to ease development. This is done by software once the system has booted for each Master of the Bus Matrix. When REMAP = 1, BMS is ignored. Refer to the Bus Matrix Section for more details.

When REMAP = 0, BMS allows the user, at ones convenience, to lay out the ROM or an external memory to 0x0. This is done via hardware at reset.

Note that Memory blocks not affected by these parameters can always be seen at their specified base addresses. The complete memory map is presented in Table 5-4 to Table 5-7.

The D940HF Bus Matrix manages a boot memory that depends on the level of the BMS pin at reset. The internal memory area mapped between address 0x0 and 0x000F FFFF is reserved for this purpose.

If BMS is detected at 1, the boot memory is the embedded ROM.

If BMS is detected at 0, the boot memory is the memory connected on the Chip Select 0 of the External Bus Interface.

5.4.2.1 BMS = 1, Boot on Embedded ROM

The system boots using the Boot Program from the embedded ROM following the steps listed below:

Checks the presence of an SD card with a boot.bin file in the main dir:

If the file is found:

- Downloads the code in the internal SRAM at 0x300000
- Executes Remap command
- Runs SD Boot code

If the file is not found, downloads the code from the SPI DataFlash®:

- Downloads the code in the internal SRAM at 0x300000
- Checks the presence of a valid code on the first six words
- Executes Remap command
- Runs DataFlash Boot code

In case no valid program is detected in the external SPI DataFlash:

  – Activates a Boot uploader enabling small monitor functionalities (read/write/run) interface with the SAM-BA® application
  – Performs an automatic detection of the communication link:

    Serial communication on a DBGU (XModem protocol)

    USB Device Port (CDC Protocol)

5.4.2.2 BMS = 0, Boot on External Memory

- Boot on slow clock (32,768 Hz)
- Boot with the default configuration for the Static Memory Controller, byte select mode, 32-bit data bus, Read/Write controlled by Chip Select, allows boot on 32-bit non-volatile memory.

The custom-programmed software must perform a complete configuration.

To speed up the boot sequence when booting at 32 kHz EBI NCS0 (BMS=0), the user must take the following steps:

1. Program the PMC (main oscillator enable or bypass mode).
2. Program and start the PLL.
3. Reprogram the SMC setup, cycle, hold, mode timings registers for NCS0 to adapt them to the new clock Peripheral Data Controller (PDC).
4. Switch the main clock to the new value.

5.5 Peripheral Data Controller (PDC)

The PDC acting as an AHB master controls the data transfer between on chip peripherals: USARTs, SPIs, SSCs, MCI, DBGU, TWIs and the on- and off-chip memories. This leaves both the processors free from the overhead related to this function.

The following list defines the PDC channel mapping:

CHANNEL22 = PDC_RX_TX to/from TWI1

CHANNEL21 = PDC_RX_TX to/from TWI0

CHANNEL20 = PDC_TX to DBGU

CHANNEL19 = PDC_TX to USART2

CHANNEL18 = PDC_TX to USART1

CHANNEL17 = PDC_TX to USART0

CHANNEL16 = PDC_TX to SPI1

CHANNEL15 = PDC_TX to SPI0

CHANNEL14 = PDC_TX to SSC3

CHANNEL13 = PDC_TX to SSC2

CHANNEL12 = PDC_TX to SSC1

CHANNEL11 = PDC_TX to SSC0

CHANNEL10 = PDC_RX from DBGU

CHANNEL9 = PDC_RX from USART2

CHANNEL8 = PDC_RX from USART1

CHANNEL7 = PDC_RX from USART0

CHANNEL6 = PDC_RX from SPI1

CHANNEL5 = PDC_RX from SPI0

CHANNEL4 = PDC_RX from SSC3

CHANNEL3 = PDC_RX from SSC2

CHANNEL2 = PDC_RX from SSC1

CHANNEL1 = PDC_RX from SSC0

CHANNEL0 = PDC_RX_TX to/from MMC

5.6 USB Host

The USB host acting as an AHB master controls the data exchange between the two USB host channels (port A and port B) and the ARM Internal RAM or the external memories.

The USB Host Port features:

- Compliance with Open HCI Rev 1.0 specification
- Compliance with USB V2.0 Full-speed and Low-speed Specification
- Supports both Low-speed 1.5 Mbps and Full-speed 12 Mbps USB devices
- Root hub integrated with two downstream USB ports
- Two embedded USB transceivers

5.7 Ethernet MAC 10/100

The Ethernet MAC acting as an AHB master controls the data exchange between the Ethernet channel and the ARM Internal RAM or the external memories.

The Ethernet MAC is the hardware implementation of the MAC sub-layer OSI reference model between the physical layer (PHY) and the logical link layer (LLC). It controls the data exchange between a host and a PHY layer according to Ethernet IEEE 802.3u data frame format. The Ethernet MAC contains the required logic and transmits and receives FIFOs for the DMA management. In addition, it is interfaced through MDIO/MDC pins for the PHY layer management. The Ethernet MAC can transfer data through the Reduced Media Independent Interface (RMII).

The aim of the interface reduction is to lower the pin count for a switch product that can be connected to multiple PHY interfaces. RMII mode specific characteristics are:

- Single clock at 50 MHz frequency
- Reduction of required control pins
- Reduction of data paths to di-bit (2-bit wide) by doubling clock frequency
- 10 Mbits/sec. and 100 Mbits/sec. data capability

The 50 MHz reference clock can be obtained either from PMC PCK0 output through PIOA16 which then goes toward both the external ETH PHY and D940HF EREFCLK pin or from an external dedicated oscillator.

5.8 MagicV JTAG

The MagicV-JTAG provides the JTAG interface to the MagicV core. It converts JTAG commands coming from a JTAG probe into AHB cycles. Acting as an AHB master it can access all MagicV memories and registers, thus allowing MagicV debug software to control the core and its resources: to upload/download data and programs and to configure functional and debug registers.

5.9 ARM System Internal RAM

ARM System internal RAM consists of a 48 Kbyte SRAM.

The internal SRAM can be accessed in Double-Word (32 bit), Word (16 bit) and Byte (8 bit) format; neither BURST nor ACCESS PROTECTION are supported.

This internal SRAM is split into 3 decoded areas:

ARM Instruction TCM. The user can map this SRAM block anywhere in the ARM926 instruction memory space by using CP15 instructions. This SRAM block is also accessible by the ARM926-D Master and by the enabled AHB Masters through the AHB bus at address 0x0010 0000.

ARM Data TCM. The user can map this SRAM block anywhere in the ARM926 data memory space by using CP15 instructions. This SRAM block is also accessible by the ARM926-D Master and by the enabled AHB Masters through the AHB bus at address 0x0020 0000.

ARM AHB MEM is only accessible by the AHB Masters.

After reset and until the Remap Command is performed, only the ARM AHB MEM (48Kbytes) is accessible through the AHB bus at address 0x00300000 by all the enabled AHB Masters.

After Remap, the ARM AHB MEM becomes also accessible by the ARM926 Instruction and the ARM926 Data Masters through the AHB bus at address 0x0 .

Of the 48 Kbyte SRAM available, the amount of memory assigned to each block is then software programmable as a multiple of 16 kB.

This configuration is defined through the dedicated Matrix Special Function Register (see Section 14.5.6).

The following table defines the possible size configurations: the ITCM possible sizes are in the second row; the DTCM possible sizes are in the second column and the ARM AHB MEM possible sizes are in the remaining cells.

Table 5-2. ARM AHB MEM configuration

		ITCM		
		0	**16**	**32**
DTCM	**0**	48	32	16
	16	32	16	na
	32	16	na	na

Note that of the three 16kB blocks that constitute the Internal SRAM, one is permanently assigned to ARM AHB MEM.

At reset, the whole memory (48kB) is assigned to ARM AHB MEM.

The memory blocks assigned to ITCM, DTCM and ARM AHB MEM decoded areas are not contiguous and when the user changes dynamically the Internal SRAM configuration through the first Bus Matrix Special Function Register (MATRIX SFR0), the new 16 Kbyte block organization may affect the previous configuration from a software point of view. The following table defines how the three 16 Kbyte blocks (called SRAM 0, 1, 2) are mapped in the four possible configurations.

Table 5-3. ARM DTCM-ITCM configuration

Decoded Area	Address	ITCM=0kB DTCM=0kB AHB=48kB	ITCM=16kB DTCM=0kB AHB=32kB	ITCM=0kB DTCM=16kB AHB=32kB	ITCM=16kB DTCM=16kB AHB=16kB
ITCM	**0x0010_0000**		SRAM 0		SRAM 0
DTCM	**0x0020_0000**			SRAM 1	SRAM 1
AHB	**0x0030_0000**	SRAM 2	SRAM 2	SRAM 2	SRAM 2
	0x0030_4000	SRAM 1	SRAM 1	SRAM 0	
	0x0030_8000	SRAM 0			

ARM performs an access to the ITCM and DTCM SRAM via their buses in a single ARM clock cycle (if 200 MHz; 5 ns); ARM accesses the ITCM SRAM and the DTCM SRAM via the D-AHB bus in two system clock cycles (if 100 MHz; 20 ns).

5.10 ARM System Internal ROM

The internal ROM is 8k x 32. The internal ROM stores the boot-loader program.

The internal ROM can be accessed only in Double-Word format (32 bit); neither BURST nor ACCESS PROTECTION are supported.

5.11 External Bus Interface (EBI)

Each enabled AHB master can access the external memory resources through the EBI. The External Bus IF incorporates the Static Memory Controller (SMC) and Synchronous Dynamic RAM controller (SDRAMC).

The EBI features:

- Eight Chip Select Lines (four via PIO lines)
- 26-bit Address Bus (four MSBs via PIO lines)
- 32-bit Data Bus
- Multiple Access Modes supported
- Byte Write Lines
- Programmable Wait State Generation
- Programmable Data Float Time
- Slow clock mode supported

5.11.1 Static Memory Controller (SMC)

The SMC gives the AHB enabled Hosts the capability to access the following external memories: SRAM, Nor-Flash, EPROM, EEPROM.

The additional NAND LOGIC also provides the SMC with the capability to interface the SmartMedia removable non-volatile memory cards and the Nand FLASH memory chips.

The additional Compact Flash logic provides the SMC with the capability to interface the Compact Flash removable non-volatile memory cards.

5.11.2 Synchronous Dynamic RAM Controller (SDRAMC)

The SDRAMC provides the interface to an external 16-bit or 32-bit SDRAM device.

The page size supports ranges from 2048 to 8192 and the columns from number 256 to 2048. It supports byte (8-bit), half-word (16-bit) and word (32-bit) accesses.

The SDRAMC supports a read or write burst length of one location. It does not support byte read/write bursts or half-word write bursts. It keeps track of the active row in each bank (avoiding precharge and active when, changing bank, the old row is accessed), thus maximizing SDRAM performance, e.g., the application may be placed in one bank and the data in the other banks. To optimize the performane it is advisable not to access different rows in the same bank.

The maximum number of SDRAM locations that can be randomly accessed without penalty cycles (precharge, active) corresponds to the device row size x the number of banks. The SDRAMC can support row size up to 2048 locations and 4 banks: hence maximum 8K locations can be accessed without penalties. Anyway, typical SDRAM row size are 512/256 locations so maximum 2K/1K locations can be accessed without penalties.

5.12 Memory Mapping

The present section describes the memory mapping of ARM9System.

Table 5-4 shows the D940HF global memory map:

Table 5-4. D940HF Global Memory Map

<table>
<tr><th rowspan="2">Start Address</th><th rowspan="2">Size (MB)</th><th colspan="7">masters</th></tr>
<tr><th>ARM9-I mst # 0</th><th>ARM9-D mst #1</th><th>PDC mst # 2</th><th>MagicV mst # 3</th><th>USB mst # 4</th><th>ETH mst # 5</th><th>m-JTAG mst # 6</th></tr>
<tr><td>0x0000 0000</td><td>256</td><td colspan="7">Internal Memories (See Table 5-6)</td></tr>
<tr><td>0x1000 0000</td><td>8 x 256</td><td colspan="6">External Memories (See Table 5-5)</td><td></td></tr>
<tr><td>0x9000 0000</td><td>6 x 256</td><td colspan="7">Undefined (Abort)</td></tr>
<tr><td>0xF000 0000</td><td>256</td><td></td><td colspan="3">Internal Peripherals (See Table 5-7)</td><td colspan="3"></td></tr>
</table>

Table 5-5 shows the external memory mapping:

Table 5-5. External Memory Map

Start Address	Size (MB)	masters						
		ARM9-I mst #0	ARM-D mst #1	PDC mst #2	MagicV mst #3	USB mst #4	ETH mst #5	m-JTAG mst #6
0x1000 0000	256	EBI NCS0: (Generic Static Memory)						
0x2000 0000	256	EBI NCS1: SMC (Generic Static Memory) or SDRAMC (1)						
0x3000 0000	256	EBI NCS2: SMC (Generic Static Memory)						
0x4000 0000	256	EBI NCS3: SMC (Generic Static Memory or SmartMedia/NAND-Flash)						
0x5000 0000	256	EBI NCS4: SMC (Generic Static Memory or Compact Flash slot 0)						
0x6000 0000	256	EBI NCS5: SMC (Generic Static Memory or Compact Flash slot 1)						
0x7000 0000	256	EBI NCS6: SMC (Generic Static Memory)						
0x8000 0000	256	EBI NCS7: SMC (Generic Static Memory)						

1. Please refer to Section 14.5.6.

Table 5-6 shows the internal memory mapping:

Table 5-6. Internal Memory Map

Start Address	Size (MB)	masters										
		ARM9-I mst # 0			ARM9-D mst # 1			PDC mst # 2	Magic V mst# 3	USB mst # 4	ETH mst # 5	m-JTAG mst # 6
		REMAP=0		REMAP=1	REMAP=0		REMAP=1					
		BMS=1	BMS=0		BMS=1	BMS=0						
0x0000 0000	1	IntROM	EBI NCS0	IntRAM C	IntROM	EBI NCS0	IntRAM C					
0x0010 0000	1				I-TCM							
0x0020 0000	1				D-TCM							
0x0030 0000	1	ARM AHB MEM										
0x0040 0000	1	IntROM										
0x0050 0000	1				USB cfg							
0x0060 0000	1				MagicV			Magic V				Magic V

Table 5-7. Internal Peripherals Map

Start Address	Size (byte)	masters						
		ARM9-I	ARM9-D	PDC	MagicV	USB	ETH	m-JTAG
0xF000 0000	40 x 16k		reserved					
0xFFFA 0000	16k		TC 0, 1, 2					
0xFFFA 4000	16k		USB DEV					
0xFFFA 8000	16k		MCI					
0xFFFA C000	16k		TWI-0					
0xFFFB 0000	16k		USART-0					
0xFFFB 4000	16k		USART-1					
0xFFFB 8000	16k		USART-2					
0xFFFB C000	16k		SSC-0					
0xFFFC 0000	16k		SSC-1					
0xFFFC 4000	16k		SSC-2					
0xFFFC 8000	16k		SPI-0					
0xFFFC C000	16k		SPI-1					
0xFFFD 0000	16k		SSC-3					
0xFFFD 4000	16k		TWI-1					
0xFFFD 8000	16k		ETH CFG					
0xFFFD C000	16k		CAN-0					
0xFFFE 0000	16k		CAN-1					
0xFFFE 4000	3 x 16k		reserved					
0xFFFF 0000	117 x 512		reserved					
0xFFFF EA00	512		SDRAMC					
0xFFFF EC00	512		SMC					
0xFFFF EE00	512		HMATRIX					
0xFFFF F000	512		AIC					
0xFFFF F200	512		DBGU					
0xFFFF F400	512		PIO A					
0xFFFF F600	512		PIO B					
0xFFFF F800	512		PIO C					
0xFFFF FA00	512		reserved					
0xFFFF FC00	256		PMC					
0xFFFF FD00	256		SYSC [1]					
0xFFFF FE00	2 x 256		reserved					

1.SYSC includes the following peripherals: RSTC, RTT, PIT, WDG.

5.13 APB Peripherals

The D940HF provides a rich set of peripherals connected to the APB bus. All enabled AHB masters can access these peripherals through the AHB-APB bridge.

5.13.1 Peripheral ID

Table 5-8 defines the Peripheral Identifiers of the D940HF. 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 5-8. Peripheral ID

Peripheral ID	Peripheral Clock Assignment	Host Clock Assignment
0		
1		
2	PIO A	
3	PIO B	
4	PIO C	
5	ETH APB	ETH AHB
6	USART-0	
7	USART-1	
8	USART-2	
9	MCI	
10	USB Device	
11	TWI-0	
12	SPI-0	
13	SPI-1	
14	SSC-0	
15	SSC-1	
16	SSC-2	
17	TIMER-0	
18	TIMER-1	
19	TIMER-2	
20		USB HOST
21	SSC-3	
22	TW1	
23	CAN-0	
24	CAN-1	
25		
26		MAGIC Core
27		
28		

Table 5-8. Peripheral ID (Continued)

Peripheral ID	Peripheral Clock Assignment	Host Clock Assignment
29		
30		
31		

5.13.2 Peripheral Multiplexing

The D940HF features three PIO controllers, PIOA, PIOB and PIOC, that multiplex the I/O lines of the peripheral set. Each PIO controller manages up to thirty-two lines. Each line can be assigned to one of the two peripheral functions, A or B. Table 5-9 to Table 5-11 define how the I/O lines of the peripherals A and B are multiplexed on the PIO Controllers. Note that some output only peripheral functions might be duplicated within the tables and are indicated with the suffixes II and III.

Table 5-9. PIO A Line Resource Mapping

<table>
<tr><th>PIO A</th><th>Periph INPUT A</th><th>Periph OUTPUT A</th><th>Periph INPUT B</th><th>Periph OUTPUT B</th></tr>
<tr><td>PIO A [0]</td><td colspan="2">SPI 0 bidir: MISO</td><td></td><td>MagicV output: M_SIRQ0</td></tr>
<tr><td>PIO A [1]</td><td colspan="2">SPI 0 bidir: MOSI</td><td></td><td>EBI: output: CFCE1 (III)</td></tr>
<tr><td>PIO A [2]</td><td colspan="2">SPI 0 bidir: CLK</td><td></td><td>EBI: output: CFCE2 (III)</td></tr>
<tr><td>PIO A [3]</td><td colspan="2">SPI 0 bidir: CS0</td><td></td><td>CAN 1: dout (III)</td></tr>
<tr><td>PIO A [4]</td><td></td><td>SPI 0 output: CS1</td><td></td><td>MagicV output: M_SIRQ2</td></tr>
<tr><td>PIO A [5]</td><td></td><td>SPI 0 output: CS2</td><td colspan="2">TIMER bidir: TIMER_OUT A0</td></tr>
<tr><td>PIO A [6]</td><td></td><td>SPI 0 output: CS3</td><td colspan="2">TIMER bidir: TIMER_OUT B1</td></tr>
<tr><td>PIO A [7]</td><td>USART 0 input: RXD</td><td></td><td></td><td>DBGU output: DTXD(III)</td></tr>
<tr><td>PIO A [8]</td><td colspan="2">USART 0 bidir: TXD</td><td></td><td>PMC output: CKOUT 1</td></tr>
<tr><td>PIO A [9]</td><td>USART 0 input: CTS</td><td></td><td></td><td>SPI 0 output: CS1 (III)</td></tr>
<tr><td>PIO A [10]</td><td></td><td>USART 0 output: RTS</td><td>TIMER input: TIMER_IN 1</td><td></td></tr>
<tr><td>PIO A [11]</td><td colspan="2">USART 0 bidir: SCK</td><td></td><td>SPI 0 output: CS2 (III)</td></tr>
<tr><td>PIO A [12]</td><td>AIC input: EXT_IRQ1
(also to MagicV)</td><td></td><td></td><td>USART 0 output: RTS (III)</td></tr>
<tr><td>PIO A [13]</td><td colspan="2">ETH bidir MDIO</td><td></td><td>MagicV output: M_SIRQ1</td></tr>
<tr><td>PIO A [14]</td><td></td><td>ETH output MDC</td><td>AIC input: EXT_IRQ2
(also to MagicV)</td><td></td></tr>
<tr><td>PIO A [15]</td><td></td><td>ETH output: FCE100</td><td>TIMER input: TIMER_IN 2</td><td></td></tr>
<tr><td>PIO A [016</td><td>ETH input: EREFCK</td><td></td><td></td><td>PMC output: CKOUT 0</td></tr>
<tr><td>PIO A [17]</td><td>ETH input: ECRSDV</td><td></td><td></td><td>EBI: output: NCS4/CFCS0 (III)</td></tr>
<tr><td>PIO A [18]</td><td>ETH input: ERX0</td><td></td><td></td><td>EBI: output: NCS5/CFCS1 (III)</td></tr>
<tr><td>PIO A [19]</td><td>ETH input: ERX1</td><td></td><td></td><td>EBI: output: NCS6 (III)</td></tr>
<tr><td>PIO A [20]</td><td>ETH input: ERXER</td><td></td><td></td><td>EBI: output: NCS7 (III)</td></tr>
<tr><td>PIO A [21]</td><td></td><td>ETH output: ETX0</td><td></td><td>TEST output: m_ck</td></tr>
<tr><td>PIO A [22]</td><td></td><td>ETH output: ETX1</td><td></td><td>TEST output: a_ck</td></tr>
</table>

Table 5-9. PIO A Line Resource Mapping (Continued)

PIO A	Periph INPUT A	Periph OUTPUT A	Periph INPUT B	Periph OUTPUT B
PIO A [23]		ETH output: ETXEN		MagicV output: M_SIRQ0 (III)
PIO A [24]	EBI input: BMS			MagicV output: M_SIRQ1 (III)
PIO A [25]	EBI input: NWAIT			USART 2 output: RTS (III)
PIO A [26]		EBI output: NCS4/CFCS0	TIMER bidir: TIMER_OUT A2	
PIO A [27]		EBI output: NCS5/CFCS1		PMC output: CKOUT 2
PIO A [28]		EBI output: NCS6		EBI output: SMOE
PIO A [29]		EBI output: NCS7		EBI output: SMWE
PIO A [30]		EBI output: CFCE1		PMC output: CKOUT 3
PIO A [31]		EBI output: CFCE2		MagicV output: M_SIRQ3

Table 5-10. PIO B Line Resource Mapping

PIO B	Periph INPUT A	Periph OUTPUT A	Periph INPUT B	Periph OUTPUT B
PIO B [0]	SSC: RD0			SPI 0 output: CS3 (III)
PIO B [1]		SSC: TD0	TIMER bidir: TIMER_OUT B0	
PIO B [2]	SSC: TF0			PMC CKOUT 0 (II)
PIO B [3]	SSC: TK0			CAN 0: dout (II)
PIO B [4]	SSC: RF0			USART 0 RTS (II)
PIO B [5]	SSC: RK0			MagicV output: M_SIRQ1 (II)
PIO B [6]	SSC: RD1			CAN 0: dout (III)
PIO B [7]		SSC: TD1	TIMER bidir: TIMER_OUT A1	
PIO B [8]	SSC: TF1			PMC CKOUT 1 (II)
PIO B [9]	SSC: TK1			SPI 1 output: CS1 (III)
PIO B [10]	SSC: RF1			USART 1 RTS (III)
PIO B [11]	SSC: RK1			EBI: A[22] (III)
PIO B [12]	SSC: RD2			EBI: A[23] (III)
PIO B [13]		SSC: TD2		MagicV output: M_SIRQ2 (II)
PIO B [14]	SSC: TF2			EBI: A[24] (III)
PIO B [15]	SSC: TK2			SPI 0 output: CS3 (II)
PIO B [016	SSC: RF2			ETH output: MDC (II)
PIO B [17]	SSC: RK2			ETH output: FCE100 (II)
PIO B [18]	SSC: RD3			EBI: A[25]-CFRNW (III)
PIO B [19]		SSC: TD3		MagicV output: M_SIRQ0 (II)
PIO B [20]	SSC: TF3			ETH output: MDC (III)
PIO B [21]	SSC: TK3			ETH output: FCE100 (III)

Table 5-10. PIO B Line Resource Mapping (Continued)

PIO B	Periph INPUT A	Periph OUTPUT A	Periph INPUT B	Periph OUTPUT B
PIO B [22]	SSC: RF3			USART 1 RTS (II)
PIO B [23]	SSC: RK3			DBGU output: DTXD (II)
PIO B [24]	TIMER input: TIMER_IN 0			MagicV output: M_MODE
PIO B [25]	AIC input: EXT_IRQ0 (also to MagicV)			USART 2 RTS (II)
PIO B [26]	CAN 0: din			SPI 1 output: CS2 (III)
PIO B [27]		CAN 0: dout		MagicV output: M_SIRQ3 (II)
PIO B [28]		EBI: A[22]		SPI 0 output: CS1 (II)
PIO B [29]		EBI: A[23]		SPI 0 output: CS2 (II)
PIO B [30]		EBI: A[24]		PMC CKOUT 2 (II)
PIO B [31]		EBI: A[25]-CFRNW		PMC CKOUT 3(II)

Table 5-11. PIO C Line Resource Mapping

PIO C	Periph INPUT A	Periph OUTPUT A	Periph INPUT B	Periph OUTPUT B
PIO C [0]	SPI 1 bi-directional: MISO			SSC: TD0 (II)
PIO C [1]	SPI 1 bi-directional: MOSI			SSC: TD1 (II)
PIO C [2]	SPI 1 bi-directional: CLK			SSC: TD2 (II)
PIO C [3]	SPI 1 bi-directional: CS0			ETH output: ETX0 (II)
PIO C [4]		SPI 1 output: CS1		ETH output: ETX1 (II)
PIO C [5]		SPI 1 output: CS2		MagicV output: M_SIRQ3 (III)
PIO C [6]		SPI 1 output: CS3		EBI: output: SMOE (III)
PIO C [7]	TWI 0 bi-directional: TWD			SSC: TD0 (III)
PIO C [8]	TWI 0 bi-directional: TWCK			SSC: TD1 (III)
PIO C [9]	USART 1 RXD			SSC: TD2 (III)
PIO C [10]	USART 1 TXD			ETH output: ETX0 (III)
PIO C [11]	USART 1 CTS			ETH output: ETX1 (III)
PIO C [12]		USART 1 RTS		SPI 1 output: CS1 (II)
PIO C [13]	USART 1 SCK			SSC: TD3 (II)
PIO C [14]	USART 2 RXD			EBI: A[22] (II)
PIO C [15]	USART 2 TXD			EBI: A[23] (II)
PIO C [16]	USART 2 CTS			EBI: A[24] (II)
PIO C [17]		USART 2 RTS		EBI: A[25]-CFRNW (II)
PIO C [18]	USART 2 SCK			SPI 1 output: CS2 (II)
PIO C [19]	TIMER bidir: TIMER_OUT B2			SPI 1 output: CS3 (II)
PIO C [20]	TWI 1 bi-directional: TWD			SSC: TD3 (III)

Table 5-11. PIO C Line Resource Mapping (Continued)

PIO C	Periph INPUT A	Periph OUTPUT A	Periph INPUT B	Periph OUTPUT B
PIO C [21]	TWI 1 bi-directional: TWCK			SPI 1 output: CS3 (III)
PIO C [22]	MCI bidir: MCCK			CAN 1: dout (II)
PIO C [23]	MCI bidir: MCCDA			MagicV output: M_SIRQ2 (III)
PIO C [24]	MCI bidir: MCDA0			EBI: SMOE (II)
PIO C [25]	MCI bidir: MCDA1			EBI: SMWE (II)
PIO C [26]	MCI bidir: MCDA2			EBI: NCS4/CFCS0 (II)
PIO C [27]	MCI bidir: MCDA3			EBI: NCS5/CFCS1 (II)
PIO C [28]	CAN 1: din			EBI: NCS6 (II)
PIO C [29]		CAN 1: dout		EBI: NCS7 (II)
PIO C [30]	DBGU input: DRXD			EBI: CFCE1 (II)
PIO C [31]		DBGU output: DTXD		EBI: CFCE2 (II)

After power up PIO_A[21] and PIO_A[22] get started linked to peripheral B to monitor the ARM clock and MagicV clock right after power-up.

After power-up PIO_A[23] gets started linked to peripheral A to avoid that the PAD pull-up lets PHY receive a HIGH level on ETH ETXEN after power-up (ETXEN from ETH is 0 after power-up).

After power-up PIO_A[26] to PIO_A[31] get started linked to peripheral A to let the EBI use all its Chip Select lines immediately after the reset.

After power-up PIO_B[28] to PIO_B[31] get started linked to peripheral A to let the EBI use all the address bus.

After power-up all other PIO lines start as inputs and as SW controlled (not linked to any peripheral).

All PIO, apart from PIO_A[24], have an embedded programmable pull-up (active after power-up).

PIO_A[24] input is internally connected to the BMS (Boot Memory Select); so it needs an external pull-up or pull-down to fix the BMS level (BMS is sampled only on reset rise).

Pads from PIO_A[25] to PIO_A[31] and from PIO_B[28] to PIO_B[31] are powered by VDDIOMP, the rest of the PIO pads are powered by VDDIOP.

5.13.3 System Controller (SYSC)

The SYSC includes the Reset Controller (RSTC) and the System Timers.

The RSTC manages all system resets: external devices reset, processors reset and peripheral reset.

The sources of reset can be: Power-On, Watch Dog, SW reset, External reset.

The System Timers features:

- One 16-bit Period Interval Timer (PIT)

- One 12-bit key-protected Watchdog Timer (WDG)
- One 20-bit Free-running Real-time Timer (RTT)

5.13.4 Power Management Controller (PMC)

The PMC features two clock sources: Slow Clock Oscillator (32.768 Hz) and Main Oscillator (8 to 20 MHz).

Two dividers, A and B, and two Phase Lock Loops, A and B, allow the generation of a wide range of frequencies either from the slow clock and/or from the main clock.

The PMC provides dedicated clocks toward: ARM926, the AHB Matrix, MagicV, MagicV Memories, the USB, the Ethernet MAC and all Peripherals.

5.13.5 Advanced Interrupt Controller (AIC)

The AIC features:

- Controls the interrupt lines (nIRQ and nFIQ) of ARM926
- Thirty-two individually maskable and vectored interrupt sources
- Programmable Edge-triggered or Level-sensitive Internal Sources
- Programmable Positive/Negative Edge-triggered or High/Low Level sensitive
- 8-level Priority Controller
- Fast Forcing: allows redirection of any normal interrupt source on the nFIQ

The following table defines the AIC interrupt mapping:

Table 5-12. AIC source mapping

Interrupt ID	Type	Peripheral
0 - FIQ	Edge/Level Negative/Positive	M_SIRQ0 from MagicV

Table 5-12. AIC source mapping

Interrupt ID	Type	Peripheral
1	Edge/Level Positive only	SYSIRQ: SDRAMC, DBGU, SYSC, PMC
2		PIO A
3		PIO B
4		PIO C
5		ETH
6		USART-0
7		USART-1
8		USART-2
9		MCI
10		USB Device
11		TWI-0
12		SPI-0
13		SPI-1
14		SSC-0
15		SSC-1
16		SSC-2
17		TIMER-0
18		TIMER-1
19		TIMER-2
20		USB Host
21		SSC-3
22		TW1
23		CAN-0
24		CAN-1
25	Edge/Level Negative/Positive	M_HALT from MagicV
26		M_SIRQ0 from MagicV
27		M_EXC from MagicV
28		END_DMA from MagicV
29		EXT_IRQ0 from PIOB25 also to MagicV SHARM_IRQ1[0]
30		EXT_IRQ1 from PIOA12 also to MagicV SHARM_IRQ1[1]
31		EXT_IRQ2 from PIOA14 also to MagicV SHARM_IRQ1[2]

5.13.6 Parallel Input/Output (PIO)

The three PIOs provide globally 96 programmable I/O Lines.

These lines are fully programmable through Set/Clear registers or linked to one of the two peripheral functions.

Each I/O Line (assigned to a peripheral or used as a general purpose I/O) provides:

- 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

5.13.7 Universal Synchronous Bus Device (USBD)

The USB Device provides communication services between an external host and the D940HF. The USB device is connected to the APB through a FIFO.

The USB Device features:

- 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
- Embedded Transceivers

5.13.8 Timer Counter (TC)

The TC consists of three 16-bit Timer Counter Channels providing 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

5.13.9 Two Wire Interface (TWI)

The D940HF provides two independent TWIs.

Each TWI interconnects components on a unique two-wire bus, made of one clock line and one data line which speed up to 400 Kbits per second, based on a byte oriented transfer format.

Each TWI is programmable in master, multi-master and slave mode with sequential or single-byte access.

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

5.13.10 Universal Synchronous Asynchronous Rx Tx (USART)

The D940HF provides three independent USARTs.

Each USART features:

- Synchronous and Asynchronous mode
- Programmable Baud Rate Generator (up to 115.2 Kbps in Asynchronous Mode and system clock frequency in Synchronous Mode)
- RS485 with driver control signal
- ISO7816, T = 0 or T = 1 Protocols for interfacing with smart cards
- IrDA modulation and demodulation
- PDC connection

5.13.11 Serial Synchronous Controller (SSC)

The D940HF provides four independent SSCs.

Each SSC provides a programmable serial synchronous communication link to be used in audio and telecom applications (CODECs in Master or Slave Modes, I2S, TDM Buses, Magnetic Card Reader, SPI, etc..).

The PDC connection allows a direct data transfer between the CODECs and either MagicV data memory or ARM internal memory or external memories.

5.13.12 Serial Peripheral Interface (SPI)

The D940HF provides two independent SPIs.

Each SPI supports the communication with serial external devices such as DataFlash, ADCs, DACs, LCD Controllers, CAN Controllers and Sensors.

Four chip selects with external decoder supports allow communication with up to 15 peripherals.

The PDC connection allows a direct data transfer between these serial devices and either MagicV data memory or ARM internal memory or external memories.

5.13.13 Debug Unit (DBGU)

The DBGU is a 2-wire UART dedicated to Debug Communication.

The DBGU TX and RX channels are associated with two PDC channels.

The Debug Unit also generates 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, allowing the handling of the DCC under interrupt control.

5.13.14 Controller Area Network (CAN)

The D940HF provides two independent CANs.

Each CAN is fully compliant with the CAN 2.0 Part A and 2.0 Part B.

The CAN supports bit/rate up to 1 Mbps.

5.13.15 Multimedia Card Interface (MCI)

The D940HF provides a MCI.

The MCI has two slots, each supporting:

– One slot for one MultiMediaCard bus (up to 30 cards) or

– One SD Memory Card

The PDC connection allows direct data transfer between these serial devices and MagicV data memory, ARM internal memory or the external memories.

5.14 ARMSystem-MagicV interface

MagicV is connected to ARM System through a master AHB IF and a slave AHB IF.

In Addition, ARM System and MagicV exchange a set of discrete lines for the cores interconnection.

The following lines go from ARM System to MagicV:

The three external interrupt input lines that go from the external pin (through PIO) to the AIC also go to the sharm_irq1[2:0] lines of MagicV (that activates MagicV Int1 internal interrupt line). When the PIO line is programmed to act as SW controlled it can be used by ARM to activate an interrupt toward MagicV.

The four internal interrupt lines that go from the SSC(0-3) to the AIC go also to sharm_irq0[3:0] lines of MagicV (that activates MagicV Int0 internal interrupt line).

The NINT line that goes from the AIC to PMC (it is the AND of the fast interrupt NFIQ and NIRQ toward ARM) goes also to sharm_irq1[3] line of MagicV (that activates Int1 MagicV internal interrupt line).

One clock line that goes from TIMER (TCOA1) to the external pin (through PIO) also goes to the arm_irq[0] line of MagicV. When the PIO line is programmed to act as SW controlled it can be used by ARM to activate an interrupt toward MagicV on Int2 internal interrupt line.

The interrupt line that goes from the SPI0 to the AIC goes also to arm_irq[1] line of MagicV on Int3 internal interrupt line.

Two clock lines go from PMC to MagicV providing MagicV main clock (Peripheral Clock[26]) and MagicV memories clock (PCLK[4] = 2x Peripheral Clock[26]).

The peripheral reset line that goes from RST CNTL to ARM peripherals goes also to MagicV.

Four generic interrupt lines M_SIRQ[3:0] go from MagicV to the external pin (through PIO). Two of these four interrupt lines (MSIRQ[1:0]) are also direct input of the AIC. The other two lines (MSIRQ[2:3]) can also be used as ARM interrupt source programming the related PIO line event detection interrupt. This implies that the MSIRQ[1:0] generates interrupts with high pulses, while MSIRQ[2:3] generates interrupts with toggling level signals.

Three dedicated interrupt lines (M_EXC, M_HALT, END_DMA) go from MagicV to AIC.

One dedicated status line (M_MODE) goes from MagicV to the external pin (through PIO).

A cross-triggering debug request line goes from MagicV Debug unit to ARM debug unit to signal a MagicV debug request event toward ARM debugger.

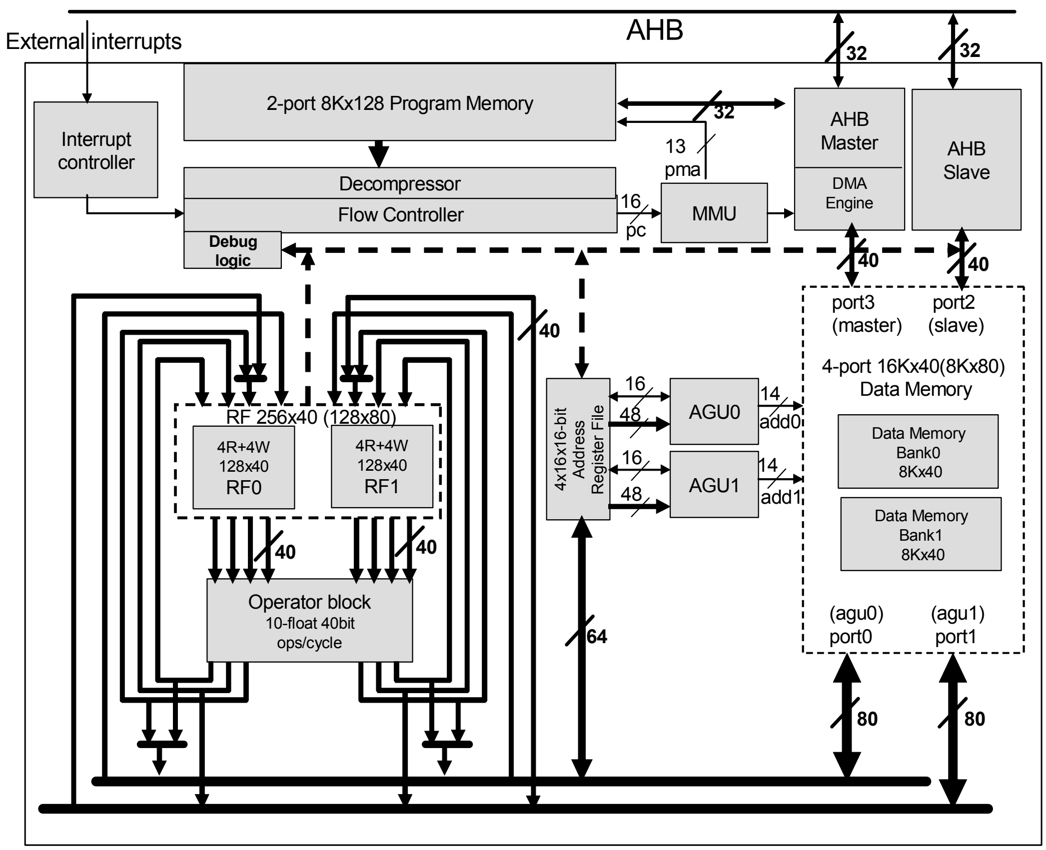
A cross-triggering debug request line goes from ARM debug unit to MagicV debug unit to signal an ARM debug request event toward MagicV debugger.

6. Magic VLIW DSP Overview

6.1 Overview

mAgicV is a high performance Very Long Instruction Word (VLIW) DSP delivering 1.0 Giga floating-point operations per second (GFLOPS) and 1.6 Gops at a clock rate of 100 MHz. It is equipped with an AHB master port and an AHB slave port for system-on-chip integration. It has 256x40-bit data registers, 16x64-bit multi-field address registers to support DSP oriented addressing modes like circular and stride accesses, 10 arithmetic operating units, two independent AGUs (Address Generation Unit) and a DMA engine. To sustain the internal parallelism, the data bandwidth through the Register File is 80 byte/cycle. The architecture is optimized to work in the complex domain. When activating all the computing units, mAgicV can produce one complete FFT butterfly per cycle. It also supports natively 2D vectorial arithmetic operations. mAgicV operates on IEEE 754 40-bit extended precision floating-point and 32-bit integer numeric format for numerical computations.

Figure 6-1. mAgicV block diagram



The Harvard memory architecture is composed of an on-chip 2x8Kx40-bit data memory and an on-chip 8Kx128-bit program memory. Efficient usage of the program memory is achieved through a mechanism of program compression, performed by the software tool chain and supported by a hardware decompression engine. A program memory management unit supports a virtual program space of 64Kx128-bit locations. Interrupts are vectorized to minimize the interrupt service latency.

6.2 VLIW overview

VLIW processors execute operations in parallel based on a fixed schedule determined when programs are compiled. Since determining the order of operations execution (including which operations can be executed simultaneously) is handled by the compiler, the processor does not need hardware support for scheduling. As a result, VLIW CPUs offer significant computational power with less hardware complexity (but greater compiler complexity) compared with most superscalar CPUs.

The rows in mAgicV program memory are 128 bit wide. When the “default” decoding scheme is applied the program word, composed of 120 bits, drives five execution units through five operation VLIW fields named issues. Eight additional bits drive the program decompression engine. The mAgicV’ issues are named: FLOW, AGU0, MUL, AGU1, ADD.

Figure 6-2. conceptual representation of issues in the default VLIW decoding scheme

FLOW	AGU0	MUL	AGU1	ADD

Two issues are associated to the pair of independent AGUs. The ADD and MUL issues drive respectively the add/subtract and the multiplier Operator units. The FLOW issue manages the program flow unit. Each issue is predicated by a predication register for conditional execution without pipeline breaking penalties.

6.3 Program Memory

The Program Memory system contains 8K*128-bit on-chip memory locations supporting up to 2 accesses/cycle. 1-accesses/cycle is reserved for the core to the fetch program, while the other access is used by the internal AHB master (i.e: DMA) or by the internal AHB slave (e.g.: debug or accesses executed by an external AHB master) accesses.

The read latency during program fetch is 1-cycle. While write and read latencies through AHB are shown on Table 6-1.

An efficient usage of the Program Memory is achieved through a program memory decompressor engine that is able to decompress, in a single clock cycle, words that are stored using a compression format. So that the total latency for program fetch, including the compression, is 2 cycles.

6.4 Register File

To provide optimal data bandwidth and to give the best support to the RISC-like programming model, mAgicV arithmetic computations are supported by a 16-port 256x40-bit entries Register File (RF). The registers are numbered from RF0 to RF255. The registers can be accessed individually for scalar operations or in pairs aligned to even addresses for operations in the complex or vectorial domain.

6.5 Operator Block

The Operator Block performs arithmetical operations. It works on 32-bit signed integers and IEEE 754 extended precision 40-bit floating-point data. 16-bit unsigned and signed integers are managed by AGUs see 16-bit. The operators are arranged in order to support:

- arithmetic on complex domain (throughput of one complex multiply, add or multiply and add per cycle);
- fast FFT (throughput of one complete butterfly computation per cycle);

- vectorial arithmetic acting on operands constituted by pairs of data.The operator block is able to launch one vectorial multiply plus one vectorial add at every cycle;
- scalar arithmetic acting on data pairs.The operator block is able to launch every cycle a pair of scalar multiply and scalar add;

The peak performance of mAgicV is achieved during single cycle FFT butterfly execution, when mAgicV delivers 10 floating-point or 32-bit signed integer operations per clock cycle.

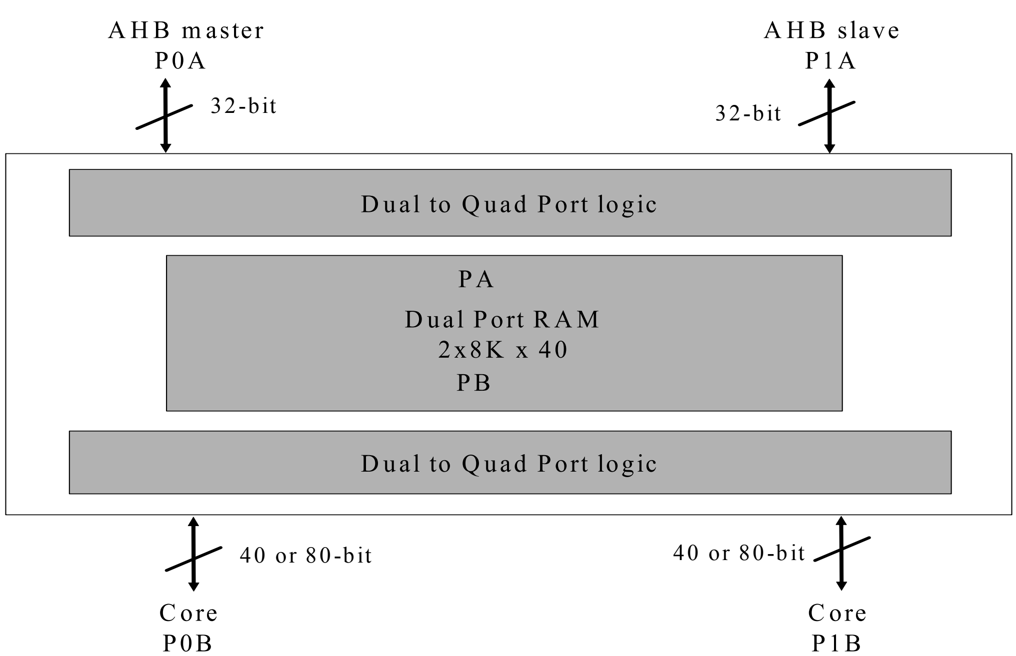
The operands manipulated by the operator block are specified by the RF addresses. The RF addresses for scalar domain operations can be odd or even. Vectorial and Complex operand pairs need even RF addresses.

6.6 On-Chip Data Memory

The Data Memory System contains 2 banks of 8K locations of 40-bit words of on-chip data memory. The On-chip Data Memory System provides a maximum throughput of 6 words/cycle. The On-chip Data Memory can be simultaneously accessed by three subjects: the computational data path, the AHB master and the AHB slave. Simultaneously, the computational datapath can fetch and store a maximum of four 40-bit data per cycle, the AHB master can drive a single access of 32-bit word per cycle x and the AHB slave can support single accesses of 32-bits per cycle. The simultaneous activity of the AHB master and slave requires an external multilayer bus matrix implementation.

Each access through P0B (and/or through P1B) can either transfer a single 40-bit data (scalar access) or access a pair of consecutive memory locations aligned to even addresses (for operation on complex or vectorial data types). Accesses through P0B and P1B are reserved to the computational data-path and their addresses are generated by AGU0 and AGU1. See Figure 6-3 for the Data Memory system.

Figure 6-3. Quad port Data Memory



6.7 Address Generation Units

There are two identical Address Generation Units in mAgicV named AGU0 and AGU1. Each AGU is driven by a dedicated VLIW issue.

The AGU can generate complex/vectorial and scalar accesses. In complex/vectorial mode two words are accesses instead of one (scalar mode). The AGU supports linear addressing and DSP oriented features like circular buffers. The address generation unit is supported by a multi field Address Registers File (ARF) composed of 4x16x16-bit registers, for a total of 64 16-bit integer registers. Register named A0-A15 are used to manage 16 bit integers/pointers, while M0-M15 registers are for 16-bit integer/pointer modifiers. When circular buffers are used, S0-S15 store the start addresses of the buffers, while L0-L15 store their lengths (zero length means no circular buffer). Each AGU contains also a private 16-bit TMP register (TMP0 and TMP1) which can be used by the AGU arithmetic and addressing operations. The AGU is able to perform 16-bit signed/unsigned integer arithmetic operations in parallel to the activities of the 40-bit floating point and 32-bit signed integer operator block.

Figure 6-4. 64-bit ARF register

63 48	47 32	31 16	15 0
S	**L**	**A**	**M**

At every clock cycle each the AGU can perform both addressing (addressing mode) and arithmetic operations (arithmetic mode).

The output of both arithmetic and addressing operations are written in the A field of an ARF register or in an internal AGU register named TMP.

The compiler, generating both addressing and arithmetic AGU operations, can exploit different solutions in terms of AGU issue generation.

The most compact and orthogonal solution is to generate issues that select a single 64-bit ARFx; but sometimes it is convenient to use the 16-bit M field from a different 64-bit ARF. When two different ARF are used, some other issues are inhibited because of the need for additional coding bits, which creates overlapping on other issues.

6.8 AHB Slave Port

AHB slave is AMBA rev 2.0 compliant, and it is directly pluggable into an AHB-lite system.

It can give only “OK” or “ERROR” responses to the AMBA AHB transactions, but it never issues a “RETRY” or a “SPLIT”.

Errors are revealed in the following cases:

1. wrong address space (address out of space or not existent)
2. data size not 32-bit (i.e. byte and half-word accesses are not permitted)
3. address not 32-bit aligned (i.e. 2 lsb need to be "00")

In case of error a pulse signal is raised and registered into the MGCEXCEPTION register.

Slave decoder receives 2 clocks, one for the AHB side and the other related to the core side.

There must be an integer ratio between the 2 clock frequencies (i.e. 1:2, 3:1, etc. etc.); skew between rising edges of clocks need to be carefully controlled and the relative phase must be stable.

See mAgicV DSP implementation manual for details on how to insert the clock-tree for the IP inside a SoC.

Slave accesses are not pipelined; each access is decoded and issued to a slave decoder block running at core frequency and when it is completed a new access can be processed. During all the processing time the AHB slave emits a “WAIT” answer.

Slave decodes three DSP addressing regions: program memory, data memory and registers, with different access times.

Table 6-1. Addressing Regions

Resource	Start Address	End Address	Size	Access	Write Latency	Read Latency
PM	0x00600000	0x0061FFFF	128KB	4 x word32	5	6
DM_I	0x00620000	0x0062FFFF	64KB	word32	5	7
DM_F	0x00640000	0x0064FFFF	64KB	word32	5	7
DM_D	0x00660000	0x0067FFFF	128KB	2 x word32	5	7
REGS	0x00680000	0x00681FFF	8KB	word32	5	6
RESERVED	0x00682000	0x006FFFFF	632KB	word32	5	6

6.9 AHB Master Port

AHB master is AMBA rev 2.0 compliant, and it is directly pluggable into an AHB system.

It does not implement protection (a default value is issued).

It supports only 32 bit accesses.

It issues only incremental bursts of unspecified length, even in case of single transfers.

It does not emit wait states.

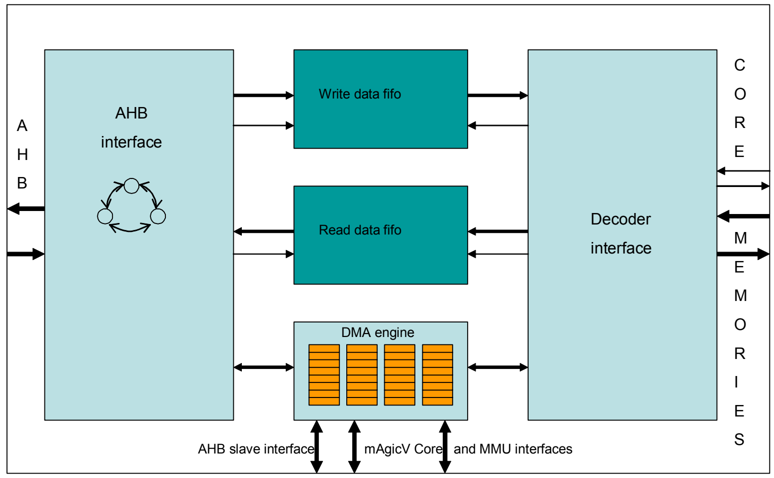
Master grant is always asserted (no arbitration is present, it's under addressed slave's responsibility the on-going of AHB transfer modulating HREADY signal)

The following picture indicates the main parts of the AHB master and the DMA engine.

When a DMA channel is ready to start a transfer it turns on the AHB master FSM for data move to/from the DSP core memories.

FIFOs are controlled by the AHB signals on one side and a decoder interface that transmits and receives data to and from memories through a master decoder block that is responsible for the correctness check and the data dispatching.

Figure 6-5. AHB master and DMA engine



AHB master is activated by a DMA engine companion.

AHB master first chooses the next winning DMA channel according to a fixed priority algorithm, then it copies transfer parameters and starts the AHB cycles as soon as possible.

A programmable length up to 64K words burst is then managed directly by the AHB master: a core engine asks for or delivers data to internal memories and the AHB bus side manages the AHB protocol. Between the AHB part and the core part there are 2 FIFOs, one for transmitting (10 locations) and the other for receiving data (16 locations).

The two sides can be clocked by different clock frequencies, but with a fixed ratio and with a fixed relative phase (ratios like 2:1, 4:1, 1:3 etc. etc. are allowed). The two different clocked worlds are separated by the FIFO.

Whenever a transfer write from the internal DSP memories to the AHB bus is running out of data the transfer is interrupted after the completion of the current data transfer and then it is continued after re-gaining bus grant, without the need for busy issuing that would occupy the bus and wast useful bandwidth.

Whenever a transfer read from the AHB bus to the DSP memories is filling up the FIFO, the transfer is interrupted after he completion of the current data transfer; the master will then transfer the FIFO content to the internal memories and only when the FIFO will be empty the transfer will continue after re-gaining bus grant, without the need for busy issuing that would waste bus cycles.FLOW Control Block

6.10 FLOW Control Block

The FLOW control block performs the following tasks:

- Registers movement

- Program flow control
- Condition management

The FLOW issue has many formats and the FLOW code can change the default format of other issues.

The basic default format of the FLOW is shown in Figure 6-6.

Figure 6-6. default FLOW issue

vliw[3:2]	vliw[118:113]	vliw[51:50]
FLOW predication	FLOW code	FLOW predication write

- **FLOW predication**

It specifies one of the four predication registers, if the condition of the pointed predication register is “false” (logic ‘0’) the issue will not be executed.

NOTE: Not all FLOW codes are predicated.

- **FLOW code**

It specifies the FLOW operations to be performed.

- **FLOW predication write**

It specifies the predication destination address.

6.11 Program Management Unit

The mAgicV architecture specifies a 16-bit virtual program memory space (64K 128-bit words). This virtual space is mapped into a physical 13-bit physical program memory space by a PMU.

The pm word (program word) is composed of 128 bit, the PMU maps 64K pm words of the external program memory in 8K pm words of the internal memory. The external program memory space is divided into 64 pages of 1K pm words. Each 1K pm word page is divided itself into sixteen chunks, each one composed of 64 pm words, as described by the following Figure.

Figure 6-7. Virtual address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
virtual page						chunk				offset					

An efficient page replacement alghorithm is realized in hardware to avoid software overhead.

It is possible to instruct the PMU to fix a set of physical pages, excluding them from the replacement algorithm. Each of the 8 physical pages has an associated PMUMAPPEDVIRT register used to specify the virtual page (each page described by one of the 64 PMUVIRT registers) and the chunks already loaded on the internal memory.

At every cycle two types of faults can be generated:

- Page fault
- Chunk fault

A page fault is generated when the virtual page isn't physically mapped into one of the eight internal physical pages. In this case the PMU finds a physical page to host the new virtual page.

If all physical pages are allocated, the PMU will replace the most recently used page with the new one, using an hardware replacement algorithm which operates on the PMU register

6.12 Data Formats

mAgicV supports the data type shown in Table 6-2.

Table 6-2. data types

type	Data width	Description
half-word	16-bits	used for signed/unsigned 16-bit integers
word	32-bits	used for signed 32-bit integers
	32-bits	used either for external memory storage of 32-bit standard precision floating-point data or for 32-bit data communication through AHB AMBA interface
extended-word	40-bits	used for internal floating point computation (extended IEEE754 format)
	64-bits	used either for external memory storage of extended precision floating-point data or for extended precision data communication through AHB AMBA interface

6.13 Data Organization

In the memory and in the RF the data is stored as 40 bit quantities (extended-word). Integers quantities have the 8 MSB padded with zero. Vector accesses occupy two consecutive addresses (a vector memory access with odd addresses generates exceptions). The “right” part of a 2-D vector quantity is contained at lower addresses.

The following figures show the representation of Table 6-2 data types.

Figure 6-8. half-word unsigned

39 … 16	15 … 0
000000000000000000000000	halfword

Figure 6-9. half-word signed extended

39 … 32	31 … 16	15 … 0
00000000	1111111111111111	halfword

Figure 6-10. word

39 … 32	31 … 0
00000000	word

A full 64-bit ARF (SLAM) register is packed in memory and in RF using two consecutive words.

Figure 6-11. even word

39 … 32	31 …	… 0
00000000	A field	M field

Figure 6-12. odd word

39 32	31	0
00000000	S field	L field

6.14 DSP States

mAgicV supports two main modes: run and debug. When the processor is in run mode there are three other possible states: step, sleep, interrupt.

Mode changes can be either caused by software control (i.e. FLOW opcodes or accesses from the external masters through the AHB slave interfaces, both writing on the MGCCTRL register), or activated by external interrupts or exceptions processing. A mode can be interrupted by a higher priority mode but never by a lower priority mode. An AHB external master can change any mAgicV state. Nested interrupts aren't supported.

Table 6-3. DSP States

Priority	State	Description
4	debug	All core pipelines are frozen, it's safe to access internal memories and registers through the AHB slave interface. Pending DMA are completed.
3	sleep	All pipeline are frozen but the state is running waiting for some events. This mode is used mainly in combination with write/read DMA operations to wait the end of the transfer (EOT).
2	step	Causes one cycle of run state followed by the debug state
1	interrupt	mAgicV executing an ISR. All pipelines are running, interrupts arriving on other lines are stored and will be served after execution of the RETI instruction. Hardware support for SW pipeline is disabled.
0	run	All pipelines are running. Interrupt will be served on branches execution.

6.15 Multicore Synchronization Support

mAgicV provides 16 mutexes to safely manage resources shared between an external AHB master controller and the mAgicV core. There is no predefined meaning for the mutex registers. The association among mutex and shared resources is driven by the software that must add control code to manage the access to the shared resources. The hardware guarantees an atomic write and test operation to lock mutexes, and a fixed priority (external AHB master first) for contemporaneous write accesses.

6.16 Event Handling

When an event occurs the execution of the instruction stream can be:

1. passed to an event handler at an address specified by one of the 8 MGCINTSVR registers.

2. resumed by a previous sleep mode.

3. halted and then pass into debug mode.

6.16.1 Interrupt handling

mAgicV allows very fast interrupt handling, treating interrupts as a routine processor instruction (branch, call, ret). Interrupts don't break pipelines and save only return program counter into the read only MGCINTRET register. mAgicV doesn't cross protection domains to take an interrupt.

Since the protection domain remains unchanged on a interrupt, the Interrupt Service Routine is called as a normal function call.

There are 8 prioritized interrupt lines. Line0 and line1 multiplex four lines each (named shared lines), so that the number of interrupt lines is 14. Each interrupt line is associated to a 16 bit interrupt vector register (MGCINTSVR) that must be set to a valid program address, corresponding to the handler interrupt routine. An interrupt, on a previously enabled and not masked interrupt line (via the MGCINTCTRL register), is registered into the PEND field of the MGCINTSTAT interrupt status register.

Interrupts can be masked using the MGCINTMASK, a masked interrupt is always registered as a pending interrupt, but it won't be served until it's masked.

When the program jumps to an Interrupt Service Routine the ISVR MGCSTAT bit will be set, indicating that no more interrupts will be served until a return from interrupt instruction (RETI) is executed. The user code return address is saved into the MGCINTRET register and it's automatically restored into the MGCPC register when a RETI issue is executed.

In case of more than one pending interrupts, the line having higher priority will be served, in case of equal priority the interrupt line with a lower number will be served.

The priority register MGCINTPRIO is a 24 bit register that allows to associate three priority bits to each line.

Pending interrupts can be set and cleared by using MGCINTSETRESET; this feature can be used to generate or clear interrupts by software over each line. Sleep and wake-up.

6.16.2 Sleep and Wakeup

mAgicV can go to sleep mode by writing the MGCCTRL register or by using the explicit FLOW codes. The processor will be waken up by one of the interrupts, or by four EOT (End of Transfer) events coming from the DMA. The events that can wake mAgicV up from a sleep state are selected using the MGCWAKECTRL control register.

6.16.3 Exceptions

mAgicV exceptions are divided into fatal and non fatal exceptions. Non masked fatal exceptions cause the processor to stop immediately and to enter into debug mode. Other exceptions can be handled in run mode by the exception interrupt routine number 6. Exception register MGCEXCEPTION collects exceptions.

6.17 Profiling Registers

The user is able to evaluate the performance of the system through two mAgicV 32 bit counter registers.

The MGCSTEP register is used to collect information on the cycles spent in run mode. It includes the cycles of pipeline stall due to program cache miss or sleep mode. This counter can be accessed by mAgicV and by an external AHB master controller.

It is possible to start and to stop the MGCSTEP counter register by accessing respectively TICKON and TICKOFF MGCCTRL control bits . An interrupt handler can be installed on INT #7 line, signalling the overflow of this counter. The overflow is registered in the MGCSTAT register and it’s cleared by write operations on the MGCSTEP register.

The PMUMISSCNT register is used to collect information about the number of programs misdone. This register can be accessed only by an external AHB master controller.

These bad events can be monitored by reading the PMUSTAT.Debug .

6.18 Debug

All the debug features can be accessed by an external AHB master that can read and write all mAgicV internal resources (memories and registers). There is a limitation on writing RF'registers.

6.18.1 Breakpoint Support

mAgicV supports breakpoints by toggling a bit of the program VLIW corresponding to the breakpoint pma. By setting PMCHKON and BREAKON on the MGCCTRL control register, a parity error is detected and interpreted as a breakpoint (MGCSTAT's PTY2BREAK flag). The external debug engine should check if the triggered breakpoint is a break point or a real parity exception.

6.18.2 Watch Point Support

mAgicV supports watchpoints through a 16 bit watch point register MGCWATCH that must contain the 16 bit internal data address of the watched variable. The watch-point logic detects write operations upon the specified watch address. The MGCCTRL's WATCHON bit must be set to enable the watchpoints.

6.18.3 Cross Triggering Support

The main function of the Cross Triggering is to pass debug events from one processor to another. The CT can communicate debug state information from one core (mAgicV) to another, so that, if required, the program execution on both processors can be stopped at the same time.

CT mode is enabled in mAgicV by setting the MGCCTRL's TRIGGON bit. In this mode a dedicated mAgicV input line (dbg_req_from_arm) is used to put mAgicV immediately (1 cycle latency) in debug mode. Vice versa mAgicV has a dedicated output line to communicate its debug state to another core(dbg_req_to_arm).

6.18.4 Step Mode Support

In this mode a program is executed step by step; this way it is possible to examine internal registers at each cycle. An external AHB master controller can activate this mode by setting the MGCCTRL's STEPON bit. The controller can advance the program execution by one cycle by setting the MGCCTRL's CONTINUE bit.

NOTE: in this mode the DMA cannot be interrupted (it continues even if the core is frozen), so that temporizations are altered compared to the normal run mode. For example, in the presence of the DMA, MGCSTEP counts less cycles than in normal run mode.

6.19 DMA

DMA engine is a single channel with 4 independent programmable set of registers.

The DMA is able to perform the following 32-bit word memory accesses:

- fixed external and/or internal address
- incremental external and/or internal address
- incremental address with a fixed external and/or internal modifier ("jump" or "stride")
- incremental address, wrapping around a specified length on external and/or internal address
- all of the above mixed
- all of the above, using the last accessed external and/or internal addresses or reloading them

All temporary conditions on the AHB bus, like loosing grant or page fault or retry/split condition, do not change the DMA channel that is currently operating (i.e. no new arbitration).

The DMA channels are serially processed and have fixed priority, the highest is channel number 3, the lowest is number 0.

Highest priority channel 3 is used by the PMU (if enabled), so the channel 3 parameters have to be always considered scratched by a user application because they are modified by the PMU. Several PMU DMA parameters (like chunck length, modifiers, external addresses) are set at bootstrap and they must be kept fixed during the program execution.

Many parameters could be fixed throughout the entire application; moreover, thanks to the possibility to redo the transfer or to continue the transfer with the same parameters and the current addresses, it could be also convenient to assign a DMA channel to a specific repetitive task, saving most of the programming costs (i.e to access peripheral registers).

NOTE: Only 32-bit word accesses are supported.

7. ARM926EJ-S Processor Overview

7.1 Overview

The ARM926EJ-S processor is a member of the ARM9™ family of general-purpose microprocessors. The ARM926EJ-S implements ARM architecture version 5TEJ and is targeted at multi-tasking applications where full memory management, high performance, low die size and low power are all important features.

The ARM926EJ-S processor supports the 32-bit ARM and 16-bit THUMB instruction sets, enabling the user to trade off between high performance and high code density. It also supports 8-bit Java instruction set and includes features for efficient execution of Java bytecode, providing a Java performance similar to a JIT (Just-In-Time compilers), for the next generation of Java-powered wireless and embedded devices. It includes an enhanced multiplier design for improved DSP performance.

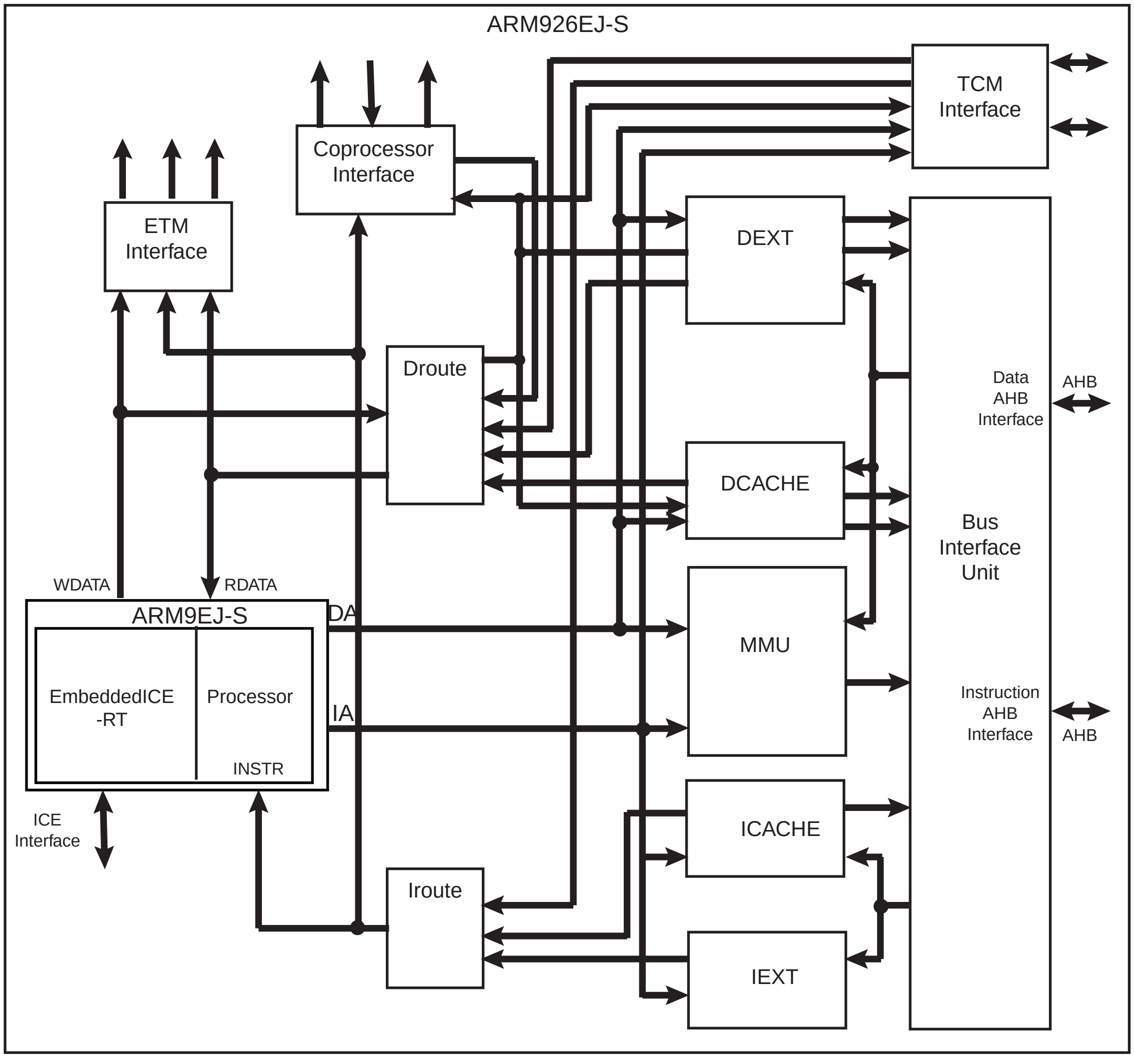
The ARM926EJ-S processor supports the ARM debug architecture and includes logic to assist in both hardware and software debug.

The ARM926EJ-S provides a complete high performance processor subsystem, including:

- an ARM9EJ-S™ integer core
- a Memory Management Unit (MMU)
- separate instruction and data AMBA™ AHB bus interfaces
- separate instruction and data TCM interfaces

7.2 Block Diagram

Figure 7-1. ARM926EJ-S Internal Functional Block Diagram



7.3 ARM9EJ-S Processor

7.3.1 ARM9EJ-S Operating States

The ARM9EJ-S processor can operate in three different states, each with a specific instruction set:

- ARM state: 32-bit, word-aligned ARM instructions.
- THUMB state: 16-bit, halfword-aligned Thumb instructions.
- Jazelle state: variable length, byte-aligned Jazelle instructions.

In Jazelle state, all instruction Fetches are in words.

7.3.2 Switching State

The operating state of the ARM9EJ-S core can be switched between:

- ARM state and THUMB state using the BX and BLX instructions, and loads to the PC

- ARM state and Jazelle state using the BXJ instruction

All exceptions are entered, handled and exited in ARM state. If an exception occurs in Thumb or Jazelle states, the processor reverts to ARM state. The transition back to Thumb or Jazelle states occurs automatically on return from the exception handler.

7.3.3 Instruction Pipelines

The ARM9EJ-S core uses two kinds of pipelines to increase the speed of the flow of instructions to the processor.

A five-stage (five clock cycles) pipeline is used for ARM and Thumb states. It consists of Fetch, Decode, Execute, Memory and Writeback stages.

A six-stage (six clock cycles) pipeline is used for Jazelle state It consists of Fetch, Jazelle/Decode (two clock cycles), Execute, Memory and Writeback stages.

7.3.4 Memory Access

The ARM9EJ-S core supports byte (8-bit), half-word (16-bit) and word (32-bit) access. Words must be aligned to four-byte boundaries, half-words must be aligned to two-byte boundaries and bytes can be placed on any byte boundary.

Because of the nature of the pipelines, it is possible for a value to be required for use before it has been placed in the register bank by the actions of an earlier instruction. The ARM9EJ-S control logic automatically detects these cases and stalls the core or forward data.

7.3.5 Jazelle Technology

The Jazelle technology enables direct and efficient execution of Java byte codes on ARM processors, providing high performance for the next generation of Java-powered wireless and embedded devices.

The new Java feature of ARM9EJ-S can be described as a hardware emulation of a JVM (Java Virtual Machine). Java mode appears as another state: instead of executing ARM or Thumb instructions, it executes Java byte codes. The Java byte code decoder logic implemented in ARM9EJ-S decodes 95% of executed byte codes and turns them into ARM instructions without any overhead, while less frequently used byte codes are broken down into optimized sequences of ARM instructions. The hardware/software split is invisible to the programmer, invisible to the application and invisible to the operating system. All existing ARM registers are re-used in Jazelle state and all registers then have particular functions in this mode.

Minimum interrupt latency is maintained across both ARM state and Java state. Since byte codes execution can be restarted, an interrupt automatically triggers the core to switch from Java state to ARM state for the execution of the interrupt handler. This means that no special provision has to be made for handling interrupts while executing byte codes, whether in hardware or in software.

7.3.6 ARM9EJ-S Operating Modes

In all states, there are seven operation modes:

- User mode is the usual ARM program execution state. It is used for executing most application programs
- Fast Interrupt (FIQ) mode is used for handling fast interrupts. It is suitable for high-speed data transfer or channel process
- Interrupt (IRQ) mode is used for general-purpose interrupt handling

- Supervisor mode is a protected mode for the operating system
- Abort mode is entered after a data or instruction prefetch abort
- System mode is a privileged user mode for the operating system
- Undefined mode is entered when an undefined instruction exception occurs

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, known as privileged modes, are entered in order to service interrupts or exceptions or to access protected resources.

7.3.7 ARM9EJ-S Registers

The ARM9EJ-S core has a total of 37 registers.

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

Table 7-1 shows all the registers in all modes.

Table 7-1. ARM9TDMI™ 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

(shaded)	Mode-specific banked registers

The ARM state register set contains 16 directly-accessible registers, r0 to r15, and an additional register, the Current Program Status Register (CPSR). Registers r0 to r13 are general-purpose registers used to hold either data or address values. Register r14 is used as a Link register that holds a value (return address) of r15 when BL or BLX is executed. Register r15 is used as a program counter (PC), whereas the Current Program Status Register (CPSR) contains condition code flags and the current mode bits.

In privileged modes (FIQ, Supervisor, Abort, IRQ, Undefined), mode-specific banked registers (r8 to r14 in FIQ mode or r13 to r14 in the other modes) become available. The corresponding banked registers r14_fiq, r14_svc, r14_abt, r14_irq, r14_und are similarly used to hold the values (return address for each mode) of r15 (PC) when interrupts and exceptions arise, or when BL or BLX instructions are executed within interrupt or exception routines. There is another register called Saved Program Status Register (SPSR) that becomes available in privileged modes instead of CPSR. This register contains condition code flags and the current mode bits saved as a result of the exception that caused entry to the current (privileged) mode.

In all modes and due to a software agreement, register r13 is used as stack pointer.

The use and the function of all the registers described above should obey ARM Procedure Call Standard (APCS) which defines:

- constraints on the use of registers
- stack conventions
- argument passing and result return

The Thumb state register set is a subset of the ARM state set. The programmer has direct access to:

- Eight general-purpose registers r0-r7
- Stack pointer, SP
- Link register, LR (ARM r14)
- PC
- CPSR

There are banked registers SPs, LRs and SPSRs for each privileged mode (for more details see the ARM9EJ-S Technical Reference Manual, ref. DDI0222B, revision r1p2 page 2-12).

7.3.7.1 *Status Registers*

The ARM9EJ-S core contains one CPSR, and five SPSRs for exception handlers to use. The program status registers:

- hold information about the most recently performed ALU operation
- control the enabling and disabling of interrupts
- set the processor operation mode

Figure 7-2. Status Register Format

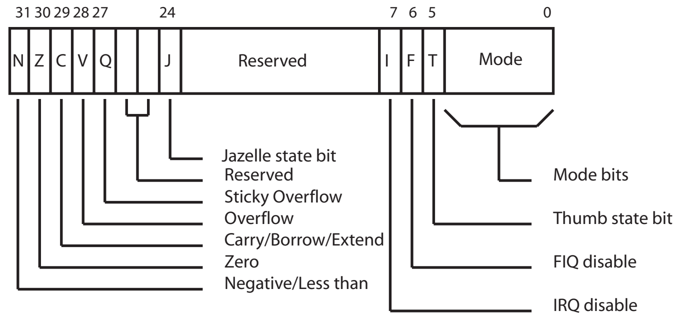


Figure 7-2 shows the status register format, where:

- N: Negative, Z: Zero, C: Carry, and V: Overflow are the four ALU flags
- The Sticky Overflow (Q) flag can be set by certain multiply and fractional arithmetic instructions like QADD, QDADD, QSUB, QDSUB, SMLAxy, and SMLAWy needed to achieve DSP operations.
  The Q flag is sticky in that, when set by an instruction, it remains set until explicitly cleared by an MSR instruction writing to the CPSR. Instructions cannot execute conditionally on the status of the Q flag.
- The J bit in the CPSR indicates when the ARM9EJ-S core is in Jazelle state, where:
  – J = 0: The processor is in ARM or Thumb state, depending on the T bit
  – J = 1: The processor is in Jazelle state.
- Mode: five bits to encode the current processor mode

7.3.7.2 Exceptions

Exception Types and Priorities

The ARM9EJ-S supports five types of exceptions. Each type drives the ARM9EJ-S in a privileged mode. The types of exceptions are:

- Fast interrupt (FIQ)
- Normal interrupt (IRQ)
- Data and Prefetched aborts (Abort)
- Undefined instruction (Undefined)
- Software interrupt and Reset (Supervisor)

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

More than one exception can happen at a time, therefore the ARM9EJ-S takes the arisen exceptions according to the following priority order:

- Reset (highest priority)
- Data Abort
- FIQ
- IRQ
- Prefetch Abort
- BKPT, Undefined instruction, and Software Interrupt (SWI) (Lowest priority)

The BKPT, or Undefined instruction, and SWI exceptions are mutually exclusive.

There is one exception in the priority scheme though, when FIQs are enabled and a Data Abort occurs at the same time as an FIQ, the ARM9EJ-S core enters the Data Abort handler, and proceeds immediately to FIQ vector. A normal return from the FIQ causes the Data Abort handler to resume execution. Data Aborts must have higher priority than FIQs to ensure that the transfer error does not escape detection.

Exception Modes and Handling

Exceptions arise whenever the normal flow of a program must be halted temporarily, for example, to service an interrupt from a peripheral.

When handling an ARM exception, the ARM9EJ-S core performs the following operations:

1. Preserves the address of the next instruction in the appropriate Link Register that corresponds to the new mode that has been entered. When the exception entry is from:
   – ARM and Jazelle states, the ARM9EJ-S copies the address of the next instruction into LR (current PC(r15) + 4 or PC + 8 depending on the exception).
   – THUMB state, the ARM9EJ-S writes the value of the PC into LR, offset by a value (current PC + 2, PC + 4 or PC + 8 depending on the exception) that causes the program to resume from the correct place on return.
2. Copies the CPSR into the appropriate SPSR.
3. Forces the CPSR mode bits to a value that depends on the exception.
4. Forces the PC to fetch the next instruction from the relevant exception vector.

The register r13 is also banked across exception modes to provide each exception handler with private stack pointer.

The ARM9EJ-S can also set the interrupt disable flags to prevent otherwise unmanageable nesting of exceptions.

When an exception has completed, the exception handler must move both the return value in the banked LR minus an offset to the PC and the SPSR to the CPSR. The offset value varies according to the type of exception. This action restores both PC and the CPSR.

The fast interrupt mode has seven private registers r8 to r14 (banked registers) to reduce or remove the requirement for register saving which minimizes the overhead of context switching.

The Prefetch Abort is one of the aborts that indicates that the current memory access cannot be completed. When a Prefetch Abort occurs, the ARM9EJ-S marks the prefetched instruction as invalid, but does not take the exception until the instruction reaches the Execute stage in the pipeline. If the instruction is not executed, for example because a branch occurs while it is in the pipeline, the abort does not take place.

The breakpoint (BKPT) instruction is a new feature of ARM9EJ-S that is destined to solve the problem of the Prefetch Abort. A breakpoint instruction operates as though the instruction caused a Prefetch Abort.
A breakpoint instruction does not cause the ARM9EJ-S to take the Prefetch Abort exception until the instruction reaches the Execute stage of the pipeline. If the instruction is not executed, for example because a branch occurs while it is in the pipeline, the breakpoint does not take place.

7.3.8 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 (bits[31:28]).

Table 7-2 gives the ARM instruction mnemonic list.

Table 7-2. ARM Instruction Mnemonic List

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

7.3.9 New ARM Instruction Set

Table 7-3. New ARM Instruction Mnemonic List

Mnemonic	Operation	Mnemonic	Operation
BXJ	Branch and exchange to Java	MRRC	Move double from coprocessor
BLX [(1)]	Branch, Link and exchange	MCR2	Alternative move of ARM reg to coprocessor
SMLAxy	Signed Multiply Accumulate 16 * 16 bit	MCRR	Move double to coprocessor
SMLAL	Signed Multiply Accumulate Long	CDP2	Alternative Coprocessor Data Processing
SMLAWy	Signed Multiply Accumulate 32 * 16 bit	BKPT	Breakpoint
SMULxy	Signed Multiply 16 * 16 bit	PLD	Soft Preload, Memory prepare to load from address
SMULWy	Signed Multiply 32 * 16 bit	STRD	Store Double
QADD	Saturated Add	STC2	Alternative Store from Coprocessor
QDADD	Saturated Add with Double	LDRD	Load Double
QSUB	Saturated subtract	LDC2	Alternative Load to Coprocessor
QDSUB	Saturated Subtract with double	CLZ	Count Leading Zeroes

Notes: 1. A Thumb BLX contains two consecutive Thumb instructions, and takes four cycles.

7.3.10 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

Table 7-4 gives the Thumb instruction mnemonic list.

Table 7-4. 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

Table 7-4. Thumb Instruction Mnemonic List (Continued)

Mnemonic	Operation	Mnemonic	Operation
LSL	Logical Shift Left	LSR	Logical Shift Right
ASR	Arithmetic Shift Right	ROR	Rotate Right
MUL	Multiply	BLX	Branch, Link, and Exchange
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
BCC	Conditional Branch	BKPT	Breakpoint

7.4 CP15 Coprocessor

Coprocessor 15, or System Control Coprocessor CP15, is used to configure and control all the items in the list below:

- ARM9EJ-S
- Caches (ICache, DCache and write buffer)
- TCM
- MMU
- Other system options

To control these features, CP15 provides 16 additional registers. See Table 7-5.

Table 7-5. CP15 Registers

Register	Name	Read/Write
0	ID Code(1)	Read/Unpredictable
0	Cache type(1)	Read/Unpredictable
0	TCM status(1)	Read/Unpredictable
1	Control	Read/write
2	Translation Table Base	Read/write
3	Domain Access Control	Read/write
4	Reserved	None
5	Data fault Status(1)	Read/write
5	Instruction fault status(1)	Read/write
6	Fault Address	Read/write
7	Cache Operations	Read/Write
8	TLB operations	Unpredictable/Write
9	Cache lockdown(2)	Read/write

Table 7-5. CP15 Registers

Register	Name	Read/Write
9	TCM region	Read/write
10	TLB lockdown	Read/write
11	Reserved	None
12	Reserved	None
13	FCSE PID(1)	Read/write
13	Context ID(1)	Read/Write
14	Reserved	None
15	Test configuration	Read/Write

Notes: 1. Register locations 0, 5 and 13 each provide access to more than one register. The register accessed depends on the value of the opcode_2 field.
2. Register location 9 provides access to more than one register. The register accessed depends on the value of the CRm field.

7.4.1 CP15 Registers Access

CP15 registers can only be accessed in privileged mode by:

- MCR (Move to Coprocessor from ARM Register) instruction is used to write an ARM register to CP15.
- MRC (Move to ARM Register from Coprocessor) instruction is used to read the value of CP15 to an ARM register.

Other instructions like CDP, LDC, STC can cause an undefined instruction exception.

The assembler code for these instructions is:

```
MCR/MRC{cond} p15, opcode_1, Rd, CRn, CRm, opcode_2.
```

The MCR, MRC instructions bit pattern is shown below:

31	30	29	28	27	26	25	24
cond				1	1	1	0

23	22	21	20	19	18	17	16
opcode_1			L	CRn			

15	14	13	12	11	10	9	8
Rd				1	1	1	1

7	6	5	4	3	2	1	0
opcode_2			1	CRm			

- **CRm[3:0]: Specified Coprocessor Action**

Determines specific coprocessor action. Its value is dependent on the CP15 register used. For details, refer to CP15 specific register behavior.

- **opcode_2[7:5]**

Determines specific coprocessor operation code. By default, set to 0.

- **Rd[15:12]: ARM Register**

Defines the ARM register whose value is transferred to the coprocessor. If R15 is chosen, the result is unpredictable.

- **CRn[19:16]: Coprocessor Register**

Determines the destination coprocessor register.

- **L: Instruction Bit**

0 = MCR instruction

1 = MRC instruction

- **opcode_1[23:20]: Coprocessor Code**

Defines the coprocessor specific code. Value is c15 for CP15.

- **cond [31:28]: Condition**

For more details, see Chapter 2 in ARM926EJ-S TRM, ref. DDI0198B.

7.5 Memory Management Unit (MMU)

The ARM926EJ-S processor implements an enhanced ARM architecture v5 MMU to provide virtual memory features required by operating systems like Symbian OS®, Windows CE®, and Linux. These virtual memory features are memory access permission controls and virtual to physical address translations.

The Virtual Address generated by the CPU core is converted to a Modified Virtual Address (MVA) by the FCSE (Fast Context Switch Extension) using the value in CP15 register13. The MMU translates modified virtual addresses to physical addresses by using a single, two-level page table set stored in physical memory. Each entry in the set contains the access permissions and the physical address that correspond to the virtual address.

The first level translation tables contain 4096 entries indexed by bits [31:20] of the MVA. These entries contain a pointer to either a 1 MB section of physical memory along with attribute information (access permissions, domain, etc.) or an entry in the second level translation tables; coarse table and fine table.

The second level translation tables contain two subtables, coarse table and fine table. An entry in the coarse table contains a pointer to both large pages and small pages along with access permissions. An entry in the fine table contains a pointer to large, small and tiny pages.

Table 7-6 shows the different attributes of each page in the physical memory.

Table 7-6. Mapping Details

Mapping Name	Mapping Size	Access Permission By	Subpage Size
Section	1M byte	Section	-
Large Page	64K bytes	4 separated subpages	16K bytes
Small Page	4K bytes	4 separated subpages	1K byte
Tiny Page	1K byte	Tiny Page	-

The MMU consists of:

- Access control logic
- Translation Look-aside Buffer (TLB)
- Translation table walk hardware

7.5.1 Access Control Logic

The access control logic controls access information for every entry in the translation table. The access control logic checks two pieces of access information: domain and access permissions. The domain is the primary access control mechanism for a memory region; there are 16 of them. It defines the conditions necessary for an access to proceed. The domain determines whether the access permissions are used to qualify the access or whether they should be ignored.

The second access control mechanism is access permissions that are defined for sections and for large, small and tiny pages. Sections and tiny pages have a single set of access permissions whereas large and small pages can be associated with 4 sets of access permissions, one for each subpage (quarter of a page).

7.5.2 Translation Look-aside Buffer (TLB)

The Translation Look-aside Buffer (TLB) caches translated entries and thus avoids going through the translation process every time. When the TLB contains an entry for the MVA (Modified Virtual Address), the access control logic determines if the access is permitted and outputs the appropriate physical address corresponding to the MVA. If access is not permitted, the MMU signals the CPU core to abort.

If the TLB does not contain an entry for the MVA, the translation table walk hardware is invoked to retrieve the translation information from the translation table in physical memory.

7.5.3 Translation Table Walk Hardware

The translation table walk hardware is a logic that traverses the translation tables located in physical memory, gets the physical address and access permissions and updates the TLB.

The number of stages in the hardware table walking is one or two depending whether the address is marked as a section-mapped access or a page-mapped access.

There are three sizes of page-mapped accesses and one size of section-mapped access. Page-mapped accesses are for large pages, small pages and tiny pages. The translation process always begins with a level one fetch. A section-mapped access requires only a level one fetch, but a page-mapped access requires an additional level two fetch. For further details on the MMU, please refer to chapter 3 in ARM926EJ-S Technical Reference Manual, ref. DDI0198B.

7.5.4 MMU Faults

The MMU generates an abort on the following types of faults:

- Alignment faults (for data accesses only)
- Translation faults
- Domain faults
- Permission faults

The access control mechanism of the MMU detects the conditions that produce these faults. If the fault is a result of memory access, the MMU aborts the access and signals the fault to the CPU core.The MMU retains status and address information about faults generated by the data accesses in the data fault status register and fault address register. It also retains the status of faults generated by instruction fetches in the instruction fault status register.

The fault status register (register 5 in CP15) indicates the cause of a data or prefetch abort, and the domain number of the aborted access when it happens. The fault address register (register 6 in CP15) holds the MVA associated with the access that caused the Data Abort. For further details on MMU faults, please refer to chapter 3 in ARM926EJ-S Technical Reference Manual, ref. DDI0198B.

7.6 Caches and Write Buffer

The ARM926EJ-S contains a 16 KB Instruction Cache (ICache), a 16 KB Data Cache (DCache), and a write buffer. Although the ICache and DCache share common features, each still has some specific mechanisms.

The caches (ICache and DCache) are four-way set associative, addressed, indexed and tagged using the Modified Virtual Address (MVA), with a cache line length of eight words with two dirty bits for the DCache. The ICache and DCache provide mechanisms for cache lockdown, cache pollution control, and line replacement.

A new feature is now supported by ARM926EJ-S caches called allocate on read-miss commonly known as wrapping. This feature enables the caches to perform critical word first cache refilling. This means that when a request for a word causes a read-miss, the cache performs an AHB access. Instead of loading the whole line (eight words), the cache loads the critical word first, so the processor can reach it quickly, and then the remaining words, no matter where the word is located in the line.

The caches and the write buffer are controlled by the CP15 register 1 (Control), CP15 register 7 (cache operations) and CP15 register 9 (cache lockdown).

7.6.1 Instruction Cache (ICache)

The ICache caches fetched instructions to be executed by the processor. The ICache can be enabled by writing 1 to I bit of the CP15 Register 1 and disabled by writing 0 to this same bit.

When the MMU is enabled, all instruction fetches are subject to translation and permission checks. If the MMU is disabled, all instructions fetches are cachable, no protection checks are made and the physical address is flat-mapped to the modified virtual address. With the MVA use disabled, context switching incurs ICache cleaning and/or invalidating.

When the ICache is disabled, all instruction fetches appear on external memory (AHB) (see Tables 4-1 and 4-2 in page 4-4 in ARM926EJ-S TRM, ref. DDI0198B).

On reset, the ICache entries are invalidated and the ICache is disabled. For best performance, ICache should be enabled as soon as possible after reset.

7.6.2 Data Cache (DCache) and Write Buffer

ARM926EJ-S includes a DCache and a write buffer to reduce the effect of main memory bandwidth and latency on data access performance. The operations of DCache and write buffer are closely connected.

7.6.2.1 DCache

The DCache needs the MMU to be enabled. All data accesses are subject to MMU permission and translation checks. Data accesses that are aborted by the MMU do not cause linefills or data accesses to appear on the AMBA AHB interface. If the MMU is disabled, all data accesses are noncachable, nonbufferable, with no protection checks, and appear on the AHB bus. All addresses are flat-mapped, VA = MVA = PA, which incurs DCache cleaning and/or invalidating every time a context switch occurs.

The DCache stores the Physical Address Tag (PA Tag) from which every line was loaded and uses it when writing modified lines back to external memory. This means that the MMU is not involved in write-back operations.

Each line (8 words) in the DCache has two dirty bits, one for the first four words and the other one for the second four words. These bits, if set, mark the associated half-lines as dirty. If the cache line is replaced due to a linefill or a cache clean operation, the dirty bits are used to decide whether all, half or none is written back to memory.

DCache can be enabled or disabled by writing either 1 or 0 to bit C in register 1 of CP15 (see Tables 4-3 and 4-4 on page 4-5 in ARM926EJ-S TRM, ref. DDI0222B).

The DCache supports write-through and write-back cache operations, selected by memory region using the C and B bits in the MMU translation tables.

The DCache contains an eight data word entry, single address entry write-back buffer used to hold write-back data for cache line eviction or cleaning of dirty cache lines.

The Write Buffer can hold up to 16 words of data and four separate addresses. DCache and Write Buffer operations are closely connected as their configuration is set in each section by the page descriptor in the MMU translation table.

7.6.2.2 Write Buffer

The ARM926EJ-S contains a write buffer that has a 16-word data buffer and a four- address buffer. The write buffer is used for all writes to a bufferable region, write-through region and write-back region. It also allows to avoid stalling the processor when writes to external memory are performed. When a store occurs, data is written to the write buffer at core speed (high speed). The write buffer then completes the store to external memory at bus speed (typically slower than the core speed). During this time, the ARM9EJ-S processor can preform other tasks.

DCache and Write Buffer support write-back and write-through memory regions, controlled by C and B bits in each section and page descriptor within the MMU translation tables.

Write-though Operation

When a cache write hit occurs, the DCache line is updated. The updated data is then written to the write buffer which transfers it to external memory.

When a cache write miss occurs, a line, chosen by round robin or another algorithm, is stored in the write buffer which transfers it to external memory.

Write-back Operation

When a cache write hit occurs, the cache line or half line is marked as dirty, meaning that its contents are not up-to-date with those in the external memory.

When a cache write miss occurs, a line, chosen by round robin or another algorithm, is stored in the write buffer which transfers it to external memory.

7.7 Tightly-Coupled Memory Interface

7.7.1 TCM Description

The ARM926EJ-S processor features a Tightly-Coupled Memory (TCM) interface, which enables separate instruction and data TCMs (ITCM and DTCM) to be directly reached by the processor. TCMs are used to store real-time and performance critical code, they also provide a DMA support mechanism. Unlike AHB accesses to external memories, accesses to TCMs are fast and deterministic and do not incur bus penalties.

The user has the possibility to independently configure each TCM size with values within the following ranges, [0 KB, 64 KB] for ITCM size and [0 KB, 64 KB] for DTCM size.

TCMs can be configured by two means: HMATRIX TCM register and TCM region register (register 9) in CP15 and both steps should be performed. HMATRIX TCM register sets TCM size whereas TCM region register (register 9) in CP15 maps TCMs and enables them.

The data side of the ARM9EJ-S core is able to access the ITCM. This is necessary to enable code to be loaded into the ITCM, for SWI and emulated instruction handlers, and for accesses to PC-relative literal pools.

7.7.2 Enabling and Disabling TCMs

Prior to any enabling step, the user should configure the TCM sizes in HMATRIX TCM register (see Section 14.5.6). Then enabling TCMs is performed by using TCM region register (register

9) in CP15. The user should use the same sizes as those put in HMATRIX TCM register. For further details and programming tips, please refer to chapter 2.3 in ARM926EJ-S TRM, ref. DDI0222B.

7.7.3 TCM Mapping

The TCMs can be located anywhere in the memory map, with a single region available for ITCM and a separate region available for DTCM. The TCMs are physically addressed and can be placed anywhere in physical address space. However, the base address of a TCM must be aligned to its size, and the DTCM and ITCM regions must not overlap. TCM mapping is performed by using TCM region register (register 9) in CP15. The user should input the right mapping address for TCMs.

7.8 Bus Interface Unit

The ARM926EJ-S features a Bus Interface Unit (BIU) that arbitrates and schedules AHB requests. The BIU implements a multi-layer AHB, based on the AHB-Lite protocol, that enables parallel access paths between multiple AHB masters and slaves in a system. This is achieved by using a more complex interconnection matrix and gives the benefit of increased overall bus bandwidth, and a more flexible system architecture.

The multi-master bus architecture has a number of benefits:

- It allows the development of multi-master systems with an increased bus bandwidth and a flexible architecture.
- Each AHB layer becomes simple because it only has one master, so no arbitration or master-to-slave muxing is required. AHB layers, implementing AHB-Lite protocol, do not have to support request and grant, nor do they have to support retry and split transactions.
- The arbitration becomes effective when more than one master wants to access the same slave simultaneously.

7.8.1 Supported Transfers

The ARM926EJ-S processor performs all AHB accesses as single word, bursts of four words, or bursts of eight words. Any ARM9EJ-S core request that is not 1, 4, 8 words in size is split into packets of these sizes. Note that the Atmel bus is AHB-Lite protocol compliant, hence it does not support split and retry requests.

Table 7-7 gives an overview of the supported transfers and different kinds of transactions they are used for.

Table 7-7. Supported Transfers

HBurst[2:0]	Description	
SINGLE	Single transfer	Single transfer of word, half word, or byte: • data write (NCNB, NCB, WT, or WB that has missed in DCache) • data read (NCNB or NCB) • NC instruction fetch (prefetched and non-prefetched) • page table walk read

Table 7-7. Supported Transfers

HBurst[2:0]	Description	
INCR4	Four-word incrementing burst	Half-line cache write-back, Instruction prefetch, if enabled. Four-word burst NCNB, NCB, WT, or WB write.
INCR8	Eight-word incrementing burst	Full-line cache write-back, eight-word burst NCNB, NCB, WT, or WB write.
WRAP8	Eight-word wrapping burst	Cache linefill

7.8.2 Thumb Instruction Fetches

All instructions fetches, regardless of the state of ARM9EJ-S core, are made as 32-bit accesses on the AHB. If the ARM9EJ-S is in Thumb state, then two instructions can be fetched at a time.

7.8.3 Address Alignment

The ARM926EJ-S BIU performs address alignment checking and aligns AHB addresses to the necessary boundary. 16-bit accesses are aligned to halfword boundaries, and 32-bit accesses are aligned to word boundaries.

8. Debug and Test

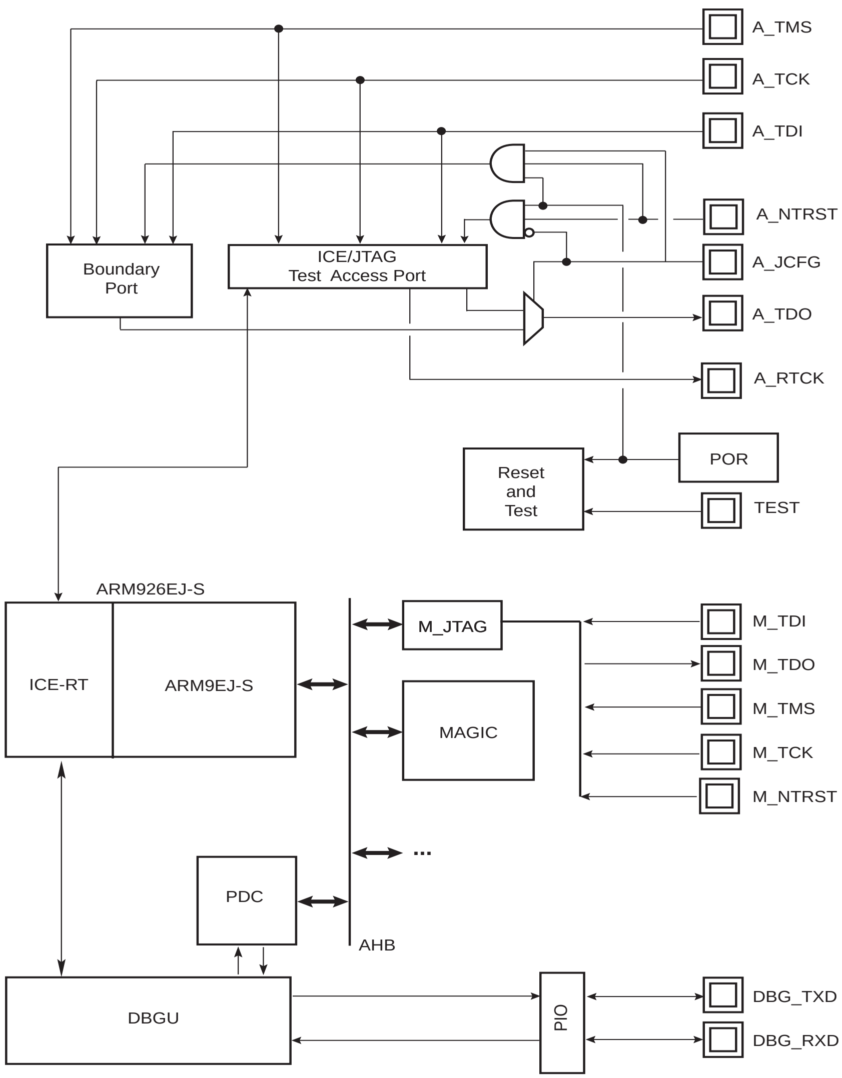
8.1 Overview

D940HF features a number of complementary debug and test capabilities. A common ARM JTAG/ICE (In-Circuit Emulator) port is used for standard ARM debugging functions, such as downloading code and single-stepping through programs. A dedicated MAGIC JTAG port provides the same functions for Magic DSP. The two JTAG ports can also interoperate featuring the crosstriggering capability. The Debug Unit provides a two-pin UART that can be used to upload an application into the internal SRAM. It manages the interrupt handling of the internal COMMTX and COMMRX signals that trace the activity of the Debug Communication Channel.

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

8.2 Block Diagram

Figure 8-1. Debug and Test Block Diagram

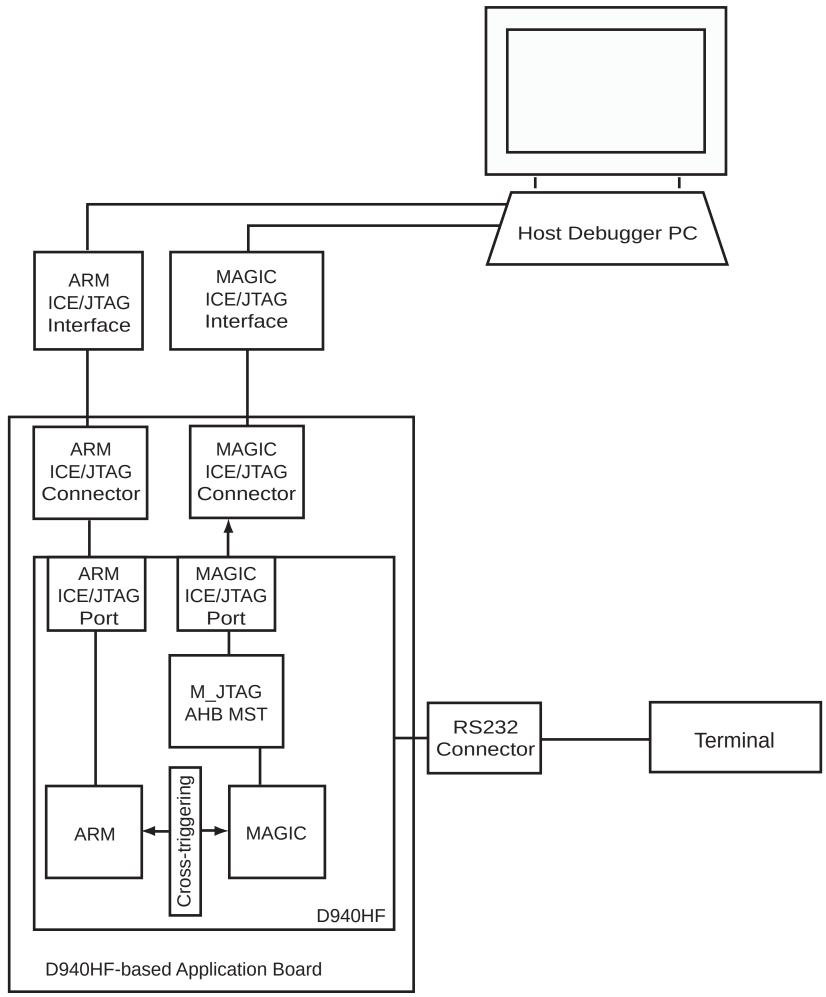


8.3 Application Examples

8.3.1 Debug Environment

Figure 8-2 on page 69 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. The Trace Port interface is used for tracing information. A software debugger running on a personal computer provides the user interface for configuring a Trace Port interface utilizing the ICE/JTAG interface.

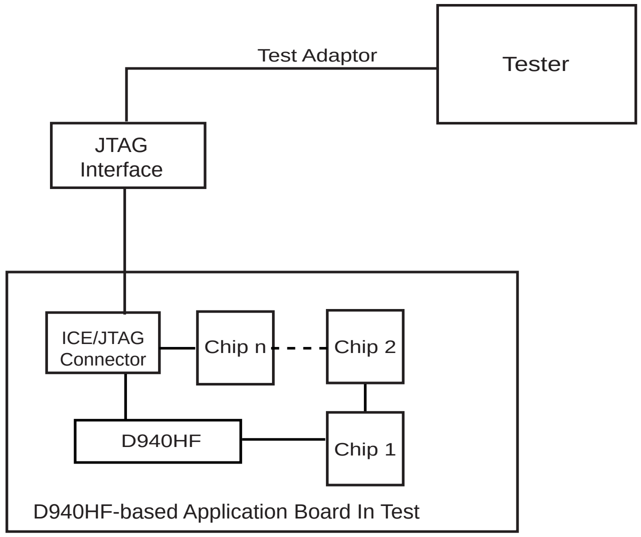
Figure 8-2. Application Debug and Trace Environment Example



8.3.2 Test Environment

Figure 8-3 on page 70 shows a test environment example. Test vectors are sent and interpreted by the tester. In this example, the “board in test” is designed using a number of JTAG-compliant devices. These devices can be connected to form a single scan chain.

Figure 8-3. Application Test Environment Example



8.4 Debug and Test Pin Description

Table 8-1. Debug and Test Pin List

Pin Name	Function	Type	Active Level
Reset/Test			
NRST	Microcontroller Reset	Input/Output	Low
TEST	Test Mode Select	Input	High
ARM-ICE and JTAG			
A_TCK	Test Clock	Input	
A_TDI	Test Data In	Input	
A_TDO	Test Data Out	Output	
A_TMS	Test Mode Select	Input	
A_NTRST	Test Reset Signal	Input	Low
A_RTCK	Returned Test Clock	Output	
A_JCFG	JTAG Selection	Input	
MAGIC-ICE			
M_TCK	Test Clock	Input	
M_TDI	Test Data In	Input	
M_TDO	Test Data Out	Output	
M_TMS	Test Mode Select	Input	
M_NTRST	Test Reset Signal	Input	Low
Debug Unit			
DRXD	Debug Receive Data	Input	
DTXD	Debug Transmit Data	Output	

8.5 Functional Description

8.5.1 Test Pin

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

8.5.2 ARM Embedded In-circuit Emulator

The ARM9EJ-S EmbeddedICE-RT™ is supported via the ICE/JTAG port. It is connected to a host computer via an ICE interface. Debug support is implemented using an ARM9EJ-S core embedded within the ARM926EJ-S. The internal state of the ARM926EJ-S is examined through an ICE/JTAG port which allows instructions to be serially inserted into the pipeline of the core without using the external data bus. Therefore, when in debug state, a store-multiple (STM) can be inserted into the instruction pipeline. This exports the contents of the ARM9EJ-S registers. This data can be serially shifted out without affecting the rest of the system.

There are two scan chains inside the ARM9EJ-S processor which support testing, debugging, and programming of the EmbeddedICE-RT. The scan chains are controlled by the ICE/JTAG port.

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

For further details on the EmbeddedICE-RT, see the ARM document ARM9EJ-S Technical Reference Manual (DDI 0222A).

8.5.3 ARM JTAG Signal Description

A_TMS is the Test Mode Select input which controls the transitions of the test interface state machine.

A_TDI is the Test Data Input line which supplies the data to the JTAG registers (Boundary Scan Register, Instruction Register, or other data registers).

A_TDO is the Test Data Output line which is used to serially output the data from the JTAG registers to the equipment controlling the test. It carries the sampled values from the boundary scan chain (or from other JTAG registers) and propagates them to the next chip in the serial test circuit.

A_NTRST (optional in IEEE Standard 1149.1) is a Test-ReSeT input which is mandatory in ARM cores and used to reset the debug logic. On Atmel ARM926EJ-S-based cores, A_NTRST is a Power On Reset output. It is asserted on power on. If necessary, the user can also reset the debug logic with the A_NTRST pin assertion during 2.5 MCK periods.

A_TCK is the Test ClocK input which enables the test interface. TCK is pulsed by the equipment controlling the test and not by the tested device. It can be pulsed at any frequency. Note that the maximum JTAG clock rate on ARM926EJ-S cores is 1/6th of the CPU clock. This gives 5.45 kHz maximum initial JTAG clock rate for an ARM9E running from the 32.768 kHz slow clock.

A_RTCK is the Return Test Clock. It is not an IEEE Standard 1149.1 signal and it is added for a better clock handling by emulators. From some ICE Interface probes, this return signal can be used to synchronize the TCK clock without caring that the given ratio between the ICE Interface clock and the system clock is equal to 1/6th. This signal is only available in JTAG ICE Mode and not in boundary scan mode.

8.5.4 MagicV In-circuit Emulator

The MagicV-Jtag block is an AHB master that provides the JTAG interface to the MagicV core.

It converts JTAG commands coming from a JTAG probe into AHB cycles.

By acting as an AHB master it can access all MagicV memories and registers, thus allowing MagicV debug software to control the core and its resources: to upload/download data and programs and to configure functional and debug registers.

8.5.5 MagicV JTAG Signal Description

M_TMS is the Test Mode Select input which controls the transitions of the test interface state machine.

M_TDI is the Test Data Input line which supplies the data to the JTAG registers (command, address and write data registers).

M_TDO is the Test Data Output line which is used to serially output the data from the JTAG registers (read data register)

M_NTRST (optional in IEEE Standard 1149.1) is a Test-ReSeT input.

M_TCK is the Test ClocK input which enables the test interface. TCK is pulsed by the equipment controlling the test and not by the tested device. It can be pulsed at any frequency.

8.5.6 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 DMA Controller channels allows 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.

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

The D940HF Debug Unit Chip ID value is 0x0E03 03E0 on 32-bit width.

8.5.7 IEEE 1149.1 JTAG Boundary Scan

IEEE 1149.1 JTAG Boundary Scan allows pin-level access independently from the device packaging technology.

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

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

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

8.5.7.1 JTAG Boundary-scan Register

The Boundary-scan Register (BSR) contains 307 bits that correspond to active pins and associated control signals.

Each D940HF input/output pin corresponds to a 2-bit register in the BSR. The INPUT/OUTPUT bit contains data that can be forced on the pad or applied to the pad to facilitate the observability. The CONTROL bit selects the direction of the pad.

Table 8-2. D940HF JTAG Boundary Scan Register

Bit Number	Pin Name	Pin Type	Associated BSR Cells
306	X32EN	INPUT	INPUT
305	ext_96m_en*	INPUT	INPUT
304	ext_96m_in*	CLOCK	CLOCK
303	PIOB0	BIDIR	CONTROL
302			IN/OUT
301	PIOB1	BIDIR	CONTROL
300			IN/OUT
299	PIOB2	BIDIR	CONTROL
298			IN/OUT
297	PIOB3	BIDIR	CONTROL
296			IN/OUT
295	PIOB4	BIDIR	CONTROL
294			IN/OUT
293	PIOB5	BIDIR	CONTROL
292			IN/OUT
291	PIOB6	BIDIR	CONTROL
290			IN/OUT
289	PIOB7	BIDIR	CONTROL
288			IN/OUT
287	PIOB8	BIDIR	CONTROL
286			IN/OUT
285	PIOB9	BIDIR	CONTROL
284			IN/OUT
283	PIOB10	BIDIR	CONTROL
282			IN/OUT
281	PIOB11	BIDIR	CONTROL
280			IN/OUT
279	PIOB12	BIDIR	CONTROL
278			IN/OUT
277	PIOB13	BIDIR	CONTROL
276			IN/OUT
275	PIOB14	BIDIR	CONTROL
274			IN/OUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
273	PIOB15	BIDIR	CONTROL
272			IN/OUT
271	PIOB16	BIDIR	CONTROL
270			IN/OUT
269	PIOB17	BIDIR	CONTROL
268			IN/OUT
267	PIOB18	BIDIR	CONTROL
266			IN/OUT
265	PIOB19	BIDIR	CONTROL
264			IN/OUT
263	PIOB28	BIDIR	CONTROL
262			IN/OUT
261	PIOB21	BIDIR	CONTROL
260			IN/OUT
259	PIOB22	BIDIR	CONTROL
258			IN/OUT
257	PIOB23	BIDIR	CONTROL
256			IN/OUT
255	PIOA13	BIDIR	CONTROL
254			IN/OUT
253	PIOA14	BIDIR	CONTROL
252			IN/OUT
251	PIOA15	BIDIR	CONTROL
250			IN/OUT
249	PIOA16	BIDIR	CONTROL
248			IN/OUT
247	PIOA17	BIDIR	CONTROL
246			IN/OUT
245	PIOA18	BIDIR	CONTROL
244			IN/OUT
243	PIOA19	BIDIR	CONTROL
242			IN/OUT
241	PIOA20	BIDIR	CONTROL
240			IN/OUT
239	PIOA21	BIDIR	CONTROL
238			IN/OUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
237	PIOA22	BIDIR	CONTROL
236			IN/OUT
235	PIOA23	BIDIR	CONTROL
234			IN/OUT
233	PIOC30	BIDIR	CONTROL
232			IN/OUT
231	PIOC31	BIDIR	CONTROL
230			IN/OUT
229	A_RTCK	OUTPUT	OUTPUT2
228	PIOC9	BIDIR	CONTROL
227			IN/OUT
226	PIOC10	BIDIR	CONTROL
225			IN/OUT
224	PIOC11	BIDIR	CONTROL
223			IN/OUT
222	PIOC12	BIDIR	CONTROL
221			IN/OUT
220	PIOC13	BIDIR	CONTROL
219			IN/OUT
218	PIOc22	BIDIR	CONTROL
217			IN/OUT
216	PIOc23	BIDIR	CONTROL
215			IN/OUT
214	PIOc24	BIDIR	CONTROL
213			IN/OUT
212	PIOc25	BIDIR	CONTROL
211			IN/OUT
210	PIOc26	BIDIR	CONTROL
209			IN/OUT
208	PIOc27	BIDIR	CONTROL
207			IN/OUT
206	TEST	INPUT	INPUT
205	NRST	BIDIR	CONTROL
204			IN/OUT
203	por_msk*	INPUT	INPUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
202	PIOA12	BIDIR	CONTROL
201			IN/OUT
200	PIOA24	BIDIR	CONTROL
199			IN/OUT
198	PIOB24	BIDIR	CONTROL
197			IN/OUT
196	PIOB25	BIDIR	CONTROL
195			IN/OUT
194	PIOC19	BIDIR	CONTROL
193			IN/OUT
192	PIOC14	BIDIR	CONTROL
191			IN/OUT
190	PIOC15	BIDIR	CONTROL
189			IN/OUT
188	PIOC16	BIDIR	CONTROL
187			IN/OUT
186	PIOC17	BIDIR	CONTROL
185			IN/OUT
184	PIOC18	BIDIR	CONTROL
183			IN/OUT
182	PIOC7	BIDIR	CONTROL
181			IN/OUT
180	PIOC8	BIDIR	CONTROL
179			IN/OUT
178	PIOC20	BIDIR	CONTROL
177			IN/OUT
176	PIOC21	BIDIR	CONTROL
175			IN/OUT
174	M_TMS	INPUT	INPUT
173	M_TCK	CLOCK	CLOCK
172	M_NTRST	INPUT	INPUT
171	M_TDI	INPUT	INPUT
170	M_TDO	OUTPUT	CONTROL
169			OUTPUT3
168	PIOA7	BIDIR	CONTROL
167			IN/OUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
166	PIOA8	BIDIR	CONTROL
165			IN/OUT
164	PIOA9	BIDIR	CONTROL
163			IN/OUT
162	PIOA10	BIDIR	CONTROL
161			IN/OUT
160	PIOA11	BIDIR	CONTROL
159			IN/OUT
158	PIOC0	BIDIR	CONTROL
157			IN/OUT
156	PIOC1	BIDIR	CONTROL
155			IN/OUT
154	PIOC2	BIDIR	CONTROL
153			IN/OUT
152	PIOC3	BIDIR	CONTROL
151			IN/OUT
150	PIOC4	BIDIR	CONTROL
149			IN/OUT
148	PIOC5	BIDIR	CONTROL
147			IN/OUT
146	PIOC6	BIDIR	CONTROL
145			IN/OUT
144	PIOC28	BIDIR	CONTROL
143			IN/OUT
142	PIOC29	BIDIR	CONTROL
141			IN/OUT
140	PIOB26	BIDIR	CONTROL
139			IN/OUT
138	PIOB27	BIDIR	CONTROL
137			IN/OUT
136	PIOA0	BIDIR	CONTROL
135			IN/OUT
134	PIOA1	BIDIR	CONTROL
133			IN/OUT
132	PIOA2	BIDIR	CONTROL
131			IN/OUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
130	PIOA3	BIDIR	CONTROL
129			IN/OUT
128	PIOA4	BIDIR	CONTROL
127			IN/OUT
126	PIOA5	BIDIR	CONTROL
125			IN/OUT
124	PIOA6	BIDIR	CONTROL
123			IN/OUT
122	PIOB28	BIDIR	CONTROL
121			IN/OUT
120	PIOB29	BIDIR	CONTROL
119			IN/OUT
118	PIOB30	BIDIR	CONTROL
117			IN/OUT
116	PIOB31	BIDIR	CONTROL
115			IN/OUT
114	PIOA25	BIDIR	CONTROL
113			IN/OUT
112	PIOA26	BIDIR	CONTROL
111			IN/OUT
110	PIOA27	BIDIR	CONTROL
109			IN/OUT
108	PIOA28	BIDIR	CONTROL
107			IN/OUT
106	PIOA29	BIDIR	CONTROL
105			IN/OUT
104	PIOA30	BIDIR	CONTROL
103			IN/OUT
102	PIOA31	BIDIR	CONTROL
101			IN/OUT
100	SD_A10	OUTPUT	OUTPUT2
99	NCS0	OUTPUT	OUTPUT2
98	NCS1	OUTPUT	OUTPUT2
97	NCS2	OUTPUT	OUTPUT2
96	NCS3	OUTPUT	OUTPUT2
95	NRD	OUTPUT	OUTPUT2

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
94	NWR0	OUTPUT	OUTPUT2
93	NWR1	OUTPUT	OUTPUT2
92	NWR3	OUTPUT	OUTPUT2
91	SDCK	OUTPUT**	CONTROL
90			IN/OUT
89	SD_CKE	OUTPUT	OUTPUT2
88	SD_NRAS	OUTPUT	OUTPUT2
87	SD_NRAS	OUTPUT	OUTPUT2
86	SD_NWE	OUTPUT	OUTPUT2
85	A0	OUTPUT	OUTPUT2
84	A1	OUTPUT	OUTPUT2
83	A2	OUTPUT	OUTPUT2
82	A3	OUTPUT	OUTPUT2
81	A4	OUTPUT	OUTPUT2
80	A5	OUTPUT	OUTPUT2
79	A6	OUTPUT	OUTPUT2
78	A7	OUTPUT	OUTPUT2
77	A8	OUTPUT	OUTPUT2
76	A9	OUTPUT	OUTPUT2
75	A10	OUTPUT	OUTPUT2
74	A11	OUTPUT	OUTPUT2
73	A12	OUTPUT	OUTPUT2
72	A13	OUTPUT	OUTPUT2
71	A14	OUTPUT	OUTPUT2
70	A15	OUTPUT	OUTPUT2
69	A16	OUTPUT	OUTPUT2
68	A17	OUTPUT	OUTPUT2
67	A18	OUTPUT	OUTPUT2
66	A19	OUTPUT	OUTPUT2
65	A20	OUTPUT	OUTPUT2
64	A21	OUTPUT	OUTPUT2
63	D0	BIDIR	CONTROL
62			IN/OUT
61	D1	BIDIR	CONTROL
60			IN/OUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
59	D2	BIDIR	CONTROL
58			IN/OUT
57	D3	BIDIR	CONTROL
56			IN/OUT
55	D4	BIDIR	CONTROL
54			IN/OUT
53	D5	BIDIR	CONTROL
52			IN/OUT
51	D6	BIDIR	CONTROL
50			IN/OUT
49	D7	BIDIR	CONTROL
48			IN/OUT
47	D8	BIDIR	CONTROL
46			IN/OUT
45	D9	BIDIR	CONTROL
44			IN/OUT
43	D10	BIDIR	CONTROL
42			IN/OUT
41	D11	BIDIR	CONTROL
40			IN/OUT
39	D12	BIDIR	CONTROL
38			IN/OUT
37	D13	BIDIR	CONTROL
36			IN/OUT
35	D14	BIDIR	CONTROL
34			IN/OUT
33	D15	BIDIR	CONTROL
32			IN/OUT
31	D16	BIDIR	CONTROL
30			IN/OUT
29	D17	BIDIR	CONTROL
28			IN/OUT
27	D18	BIDIR	CONTROL
26			IN/OUT
25	D19	BIDIR	CONTROL
24			IN/OUT

Table 8-2. D940HF JTAG Boundary Scan Register (Continued)

Bit Number	Pin Name	Pin Type	Associated BSR Cells
23	D20	BIDIR	CONTROL
22			IN/OUT
21	D21	BIDIR	CONTROL
20			IN/OUT
19	D22	BIDIR	CONTROL
18			IN/OUT
17	D23	BIDIR	CONTROL
16			IN/OUT
15	D24	BIDIR	CONTROL
14			IN/OUT
13	D25	BIDIR	CONTROL
12			IN/OUT
11	D26	BIDIR	CONTROL
10			IN/OUT
9	D27	BIDIR	CONTROL
8			IN/OUT
7	D28	BIDIR	CONTROL
6			IN/OUT
5	D29	BIDIR	CONTROL
4			IN/OUT
3	D30	BIDIR	CONTROL
2			IN/OUT
1	D31	BIDIR	CONTROL
0			IN/OUT

*: Only for production/test purposes.

**: Bidir only for internal loopback; it acts exclusively as an output.

9. Boot Program

9.1 Description

The Boot Program integrates different programs that manage download and/or upload into the different memories of the product.

First, it initializes the Debug Unit serial port (DBGU) and the USB Device Port. Then the SD Card Boot program is executed. It looks for a boot.bin file in the root directory of a FAT12/16/32 formatted SD Card.

If such a file is found, the code is downloaded into the internal SRAM. This is followed by a remap and a jump to the first adddress of the SRAM.

If the SD Card is not formatted or if boot.bin file is not found, the DataFlash® Boot program is executed.

It looks for a sequence of seven valid ARM exception vectors in a DataFlash connected to the SPI. All these vectors must be B-branch or LDR load register instructions except for the sixth vector. This vector is used to store the size of the image to download.

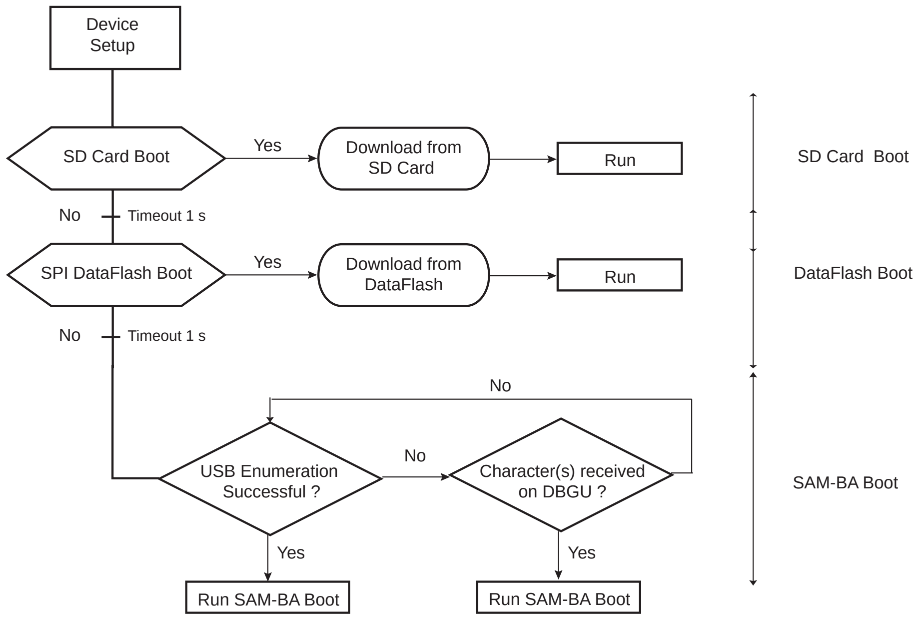
If a valid sequence is found, the code is downloaded into the internal SRAM. This is followed by a remap and a jump to the first address of the SRAM.

If no valid ARM vector sequence is found, SAM-BA Boot is then executed. It waits for transactions either on the USB device, or on the DBGU serial port.

9.2 Flow Diagram

The Boot Program implements the algorithm in Figure 9-1.

Figure 9-1. Boot Program Algorithm Flow Diagram



9.3 Device Initialization

Initialization steps are described below:

1. Stack setup for ARM supervisor mode
2. Main Oscillator Frequency Detection
3. C variable initialization
4. PLL setup: PLLB is enabled to generate a 48 MHz clock necessary to use the USB Device.
5. Initialization of the DBGU serial port (115200 bauds, 8, N, 1)
6. Disable the Watchdog and enable the user reset
7. Initialization of the USB Device Port
8. Jump to SD Card Boot sequence

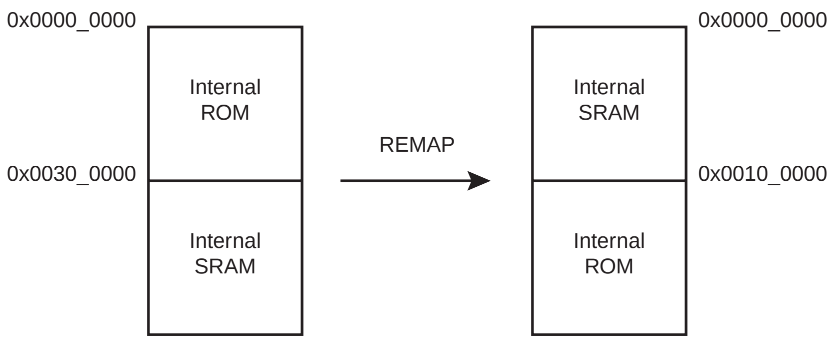
If SD Card Boot fails:

9. Jump to DataFlash Boot sequence

If DataFlash Boot fails:

10. Activation of the Instruction Cache
11. Jump to SAM-BA Boot sequence

Figure 9-2. Remap Action after Download Completion



9.4 SD Card Boot

The SD Card Boot program searches for a valid application in the SD Card memory. It looks for a boot.bin file in the root directory of a FAT12/16/32 formatted SD Card. If a valid file is found, this application is loaded into the internal SRAM and executed by branching at address 0x0000_0000 after remap. This application may be the application code or a second-level bootloader.

9.5 DataFlash Boot

The DataFlash Boot program searches for a valid application in the SPI DataFlash memory. If a valid application is found, this application is loaded into the internal SRAM and executed by branching at address 0x0000_0000 after remap. This application may be the application code or a second-level bootloader.

All the calls to functions are PC relative and do not use absolute addresses.

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

```
400000  ea000006  B    0x20          00  ea000006  B    0x20
400004  eaffffe   B    0x04          04  eaffffe   B    0x04
400008  ea00002f  B    _main         08  ea00002f  B    _main
40000c  eaffffe   B    0x0c          0c  eaffffe   B    0x0c
400010  eaffffe   B    0x10          10  eaffffe   B    0x10
400014  eaffffe   B    0x14          14  eaffffe   B    0x14
400018  eaffffe   B    0x18          18  eaffffe   B    0x18
```

9.5.1 Valid Image Detection

The DataFlash Boot software looks for a valid application by analyzing the first 28 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 0x14, contains the size of the image to download. The user must replace this vector with his/her own vector (see "Structure of ARM Vector 6" on page 85).

Figure 9-3. LDR Opcode

31			28	27			24	23			20	19 ... 16	15 ... 12	11 ... 0
1	1	1	0	1	1	I	P	U	1	W	0	Rn	Rd	

Figure 9-4. B Opcode

31			28	27			24	23 ... 0
1	1	1	0	1	0	1	0	Offset (24 bits)

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

9.5.2 Structure of ARM Vector 6

The ARM exception vector 6 is used to store the information needed by the DataFlash boot program. This information is described below.

Figure 9-5. Structure of the ARM Vector 6

31	0
Size of the code to download in bytes	

9.5.2.1 Example

An example of valid vectors follows:

```
00   ea000006  B     0x20
04   eafffffe  B     0x04
08   ea00002f  B     _main
0c   eafffffe  B     0x0c
10   eafffffe  B     0x10
14   00001234  B     0x14    <- Code size = 4660 bytes
18   eafffffe  B     0x18
```

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/her application.

9.5.3 DataFlash Boot Sequence

The DataFlash boot program performs device initialization followed by the download procedure.

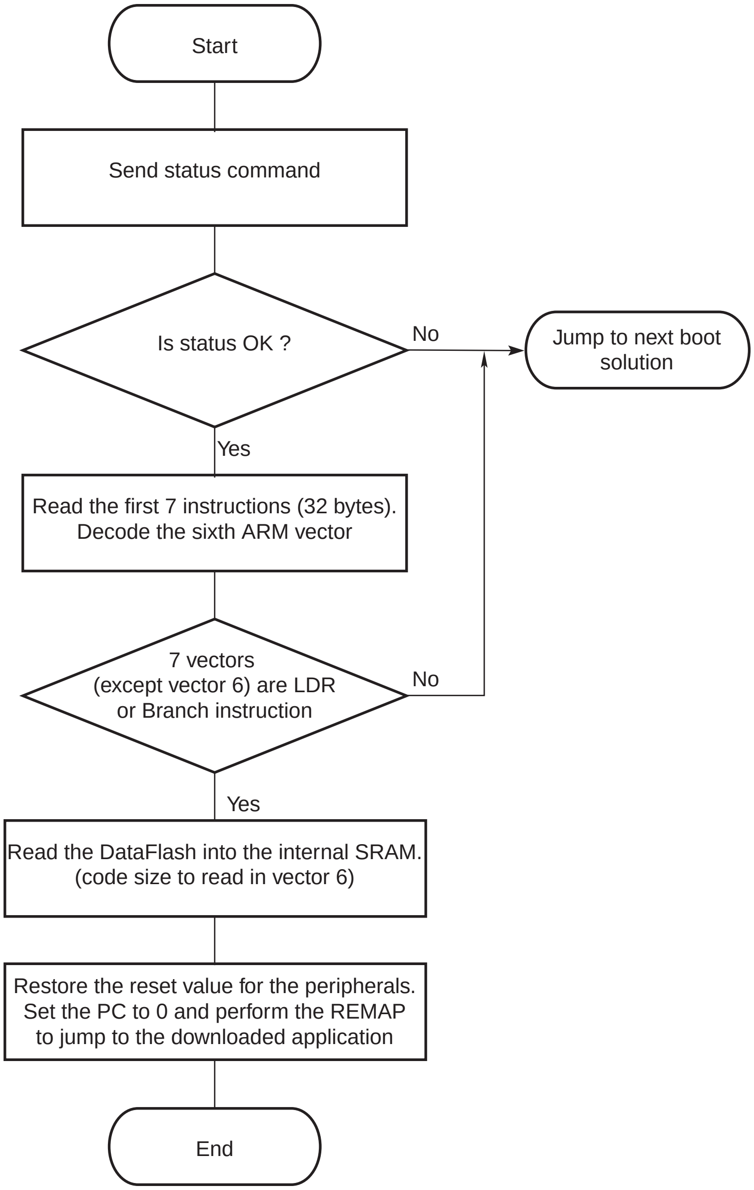
The DataFlash boot program supports all Atmel DataFlash devices. Table 9-1 summarizes the parameters to include in the ARM vector 6 for all devices.

Table 9-1. DataFlash Device

Device	Density	Page Size (bytes)	Number of Pages
AT45DB011	1 Mbit	264	512
AT45DB021	2 Mbits	264	1024
AT45DB041	4 Mbits	264	2048
AT45DB081	8 Mbits	264	4096
AT45DB161	16 Mbits	528	4096
AT45DB321	32 Mbits	528	8192
AT45DB642	64 Mbits	1056	8192
AT45DB1282	128 Mbits	1056	16384
AT45DB2562	256 Mbits	2112	16384
AT45DB5122	512 Mbits	2112	32768

The DataFlash has a Status Register that determines all the parameters required to access the device. The DataFlash boot is configured to be compatible with the future design of the DataFlash.

Figure 9-6. Serial DataFlash Download



9.6 SAM-BA Boot

If no valid DataFlash device has been found during the DataFlash boot sequence, the SAM-BA boot program is performed.

The SAM-BA boot principle is to:

– Check if the USB Device enumeration has occured.

– Check if the characters have been received on the DBGU.

– Once the communication interface is identified, the application runs in an infinite loop waiting for different commands as in Table 9-2.

Table 9-2. Commands available through the SAM-BA Boot

Command	Action	Argument(s)	Example
O	write a byte	Address, Value#	**O**200001,CA#
o	read a byte	Address,#	**o**200001,#
H	write a half word	Address, Value#	**H**200002,CAFE#
h	read a half word	Address,#	**h**200002,#
W	write a word	Address, Value#	**W**200000,CAFEDECA#
w	read a word	Address,#	**w**200000,#
S	send a file	Address,#	**S**200000,#
R	receive a file	Address, NbOfBytes#	**R**200000,1234#
G	go	Address#	**G**200200#
V	display version	No argument	**V**#

- Write commands: Write a byte (**O**), a halfword (**H**) or a word (**W**) to the target.
  – *Address*: Address in hexadecimal.
  – *Value*: Byte, halfword or word to write in hexadecimal.
  – *Output*: '>'.
- Read commands: Read a byte (**o**), a halfword (**h**) or a word (**w**) from the target.
  – *Address*: Address in hexadecimal
  – *Output*: The byte, halfword or word read in hexadecimal following by '>'
- Send a file (**S**): Send a file to a specified address
  – *Address*: Address in hexadecimal
  – *Output*: '>'.

Note: There is a time-out on this command which is reached when the prompt '>' appears before the end of the command execution.

- Receive a file (**R**): Receive data into a file from a specified address
  – *Address*: Address in hexadecimal
  – *NbOfBytes*: Number of bytes in hexadecimal to receive
  – *Output*: '>'
- Go (**G**): Jump to a specified address and execute the code
  – *Address*: Address to jump in hexadecimal
  – *Output*: '>'
- Get Version (**V**): Return the SAM-BA boot version
  – *Output*: '>'

9.6.1 DBGU Serial Port

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

The Send and Receive File commands use the Xmodem protocol to communicate. Any terminal performing this protocol can be used to send the application file to the target. The size of the binary file to send depends on the SRAM size embedded in the product. In all cases, the size of the binary file must be lower than the SRAM size because in order to work the Xmodem protocol requires some SRAM memory.

9.6.2 Xmodem Protocol

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

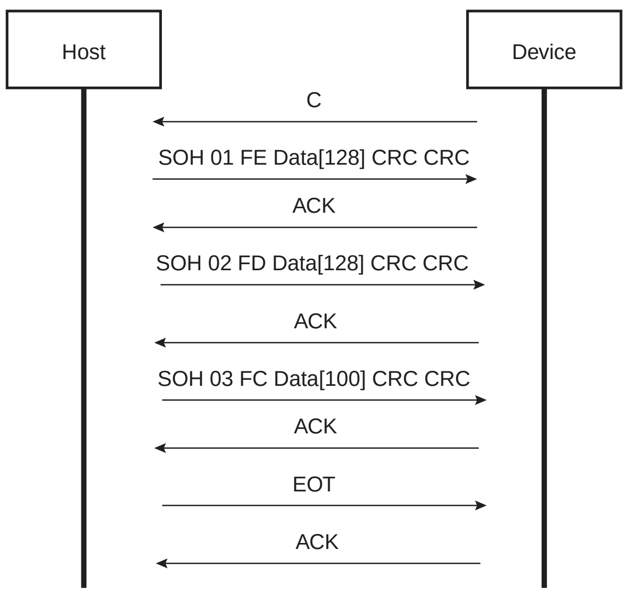
Xmodem protocol with CRC is accurate provided that both sender and receiver report successful transmission. Each block of the transfer looks like:

<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 9-7 shows a transmission using this protocol.

Figure 9-7. Xmodem Transfer Example



9.6.3 USB Device Port

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

The device uses the USB communication device class (CDC) drivers to take advantage of the installed PC RS-232 software tocommunicate through the USB. The CDC class is implemented in all Windows® releases, from Windows® 98SE to Windows XP®. The CDC document, available at www.usb.org, describes how to implement devices such as ISDN modems and virtual COM ports.

The Vendor ID is Atmel's vendor ID 0x03EB. The product ID is 0x6124. These references are used by the host operating system to mount the correct driver. On Windows systems, the INF files contain the correspondence between vendor ID and product ID.

Atmel provides an INF example to see the device as a new serial port and also provides another custom driver used by the SAM-BA application: atm6124.sys. Refer to the document "USB Basic Application", literature number 6123, for more details.

9.6.3.1 *Enumeration Process*

The USB protocol is a master/slave protocol. This is the host that starts the enumeration by sending requests to the device through the control endpoint. The device handles standard requests as defined in the USB Specification.

Table 9-3. Handled Standard Requests

Request	Definition
GET_DESCRIPTOR	Returns the current device configuration value.
SET_ADDRESS	Sets the device address for all future device access.
SET_CONFIGURATION	Sets the device configuration.
GET_CONFIGURATION	Returns the current device configuration value.
GET_STATUS	Returns status for the specified recipient.
SET_FEATURE	Used to set or enable a specific feature.
CLEAR_FEATURE	Used to clear or disable a specific feature.

The device also handles some class requests defined in the CDC class.

Table 9-4. Handled Class Requests

Request	Definition
SET_LINE_CODING	Configures DTE rate, stop bits, parity and number of character bits.
GET_LINE_CODING	Requests current DTE rate, stop bits, parity and number of character bits.
SET_CONTROL_LINE_STATE	RS-232 signal tells the DCE device that the DTE device is now present.

Unhandled requests are STALLed.

9.6.3.2 *Communication Endpoints*

There are two communication endpoints and endpoint 0 is used for the enumeration process. Endpoint 1 is a 64-byte Bulk OUT endpoint and endpoint 2 is a 64-byte Bulk IN endpoint. SAM-BA Boot commands are sent by the host through the endpoint 1. If required, the message is split by the host into several data payloads through the host driver.

If the command requires a response, the host can send IN transactions to pick up the response.

9.7 Hardware and Software Constraints

- The DataFlash and the SD Card downloaded code size must be inferior to 40 K bytes.

- The code is always downloaded from the device address 0x0000_0000 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 DataFlash must be connected to NPCS0 of the SPI.

The SPI and MCI drivers use several PIOs in alternate functions to communicate with the 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 output pins and the connected devices may occur.

It is recommended to plug in critical devices to other pins to ensure correct functionality.

Table 9-5 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 less than 1 second if no correct boot program is found.

For the DataFlash driven by the SPCK signal at 8 MHz, the time to download 40 K bytes is reduced to 68 ms.

For the SD Card driven by the MCCK signal at 12 MHz the time to download 40 K bytes is reduced to 6.8 ms.

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

Table 9-5. Pins Driven during Boot Program Execution

Peripheral	Pin	PIO Line
SPI0	MOSI	PIOA1
SPI0	MISO	PIOA0
SPI0	SPCK	PIOA2
SPI0	NPCS0	PIOA3
DBGU	DRXD	PIOA9
DBGU	DTXD	PIOA10
MCI	MCCK	PIOC22
MCI	MCCDA	PIOC23
MCI	MCDA0	PIOC24
MCI	MCDA1	PIOC25
MCI	MCDA2	PIOC26
MCI	MCDA3	PIOC27

10. Reset Controller (RSTC)

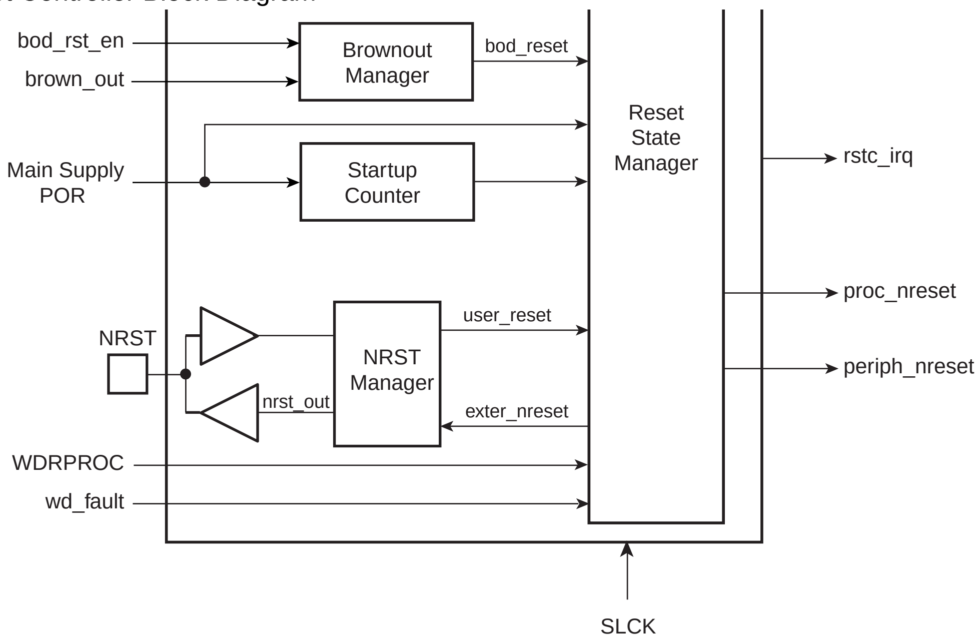
10.1 Description

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

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

10.2 Block Diagram

Figure 10-1. Reset Controller Block Diagram



10.3 Functional Description

10.3.1 Reset Controller Overview

The Reset Controller is made up of an NRST Manager, a Startup Counter and a Reset State Manager. It runs at Slow Clock and generates the following reset signals:

- proc_nreset: Processor reset line. It also resets the Watchdog Timer.
- periph_nreset: Affects the whole set of embedded peripherals.
- nrst_out: Drives the NRST pin.

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

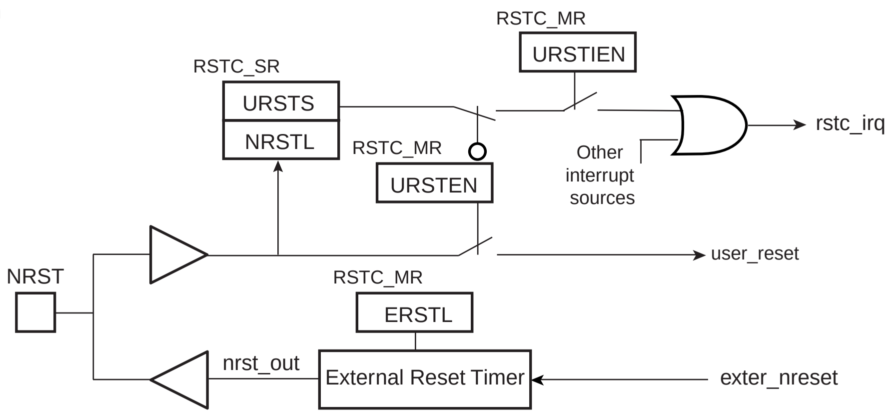
The NRST Manager shapes the NRST assertion during a programmable time, thus controlling external device resets.

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

10.3.2 NRST Manager

The NRST Manager samples the NRST input pin and drives this pin low when required by the Reset State Manager. Figure 10-2 shows the block diagram of the NRST Manager.

Figure 10-2. NRST Manager



10.3.2.1 NRST Signal or Interrupt

The NRST Manager samples the NRST pin at Slow Clock speed. When the line is detected low, a User Reset is reported to the Reset State Manager.

However, the NRST Manager can be programmed to not trigger a reset when an assertion of NRST occurs. Writing the bit URSTEN at 0 in RSTC_MR disables the User Reset trigger.

The level of the pin NRST can be read at any time in the bit NRSTL (NRST level) in RSTC_SR. As soon as the pin NRST is asserted, the bit URSTS in RSTC_SR is set. This bit clears only when RSTC_SR is read.

The Reset Controller can also be programmed to generate an interrupt instead of generating a reset. To do so, the bit URSTIEN in RSTC_MR must be written at 1.

10.3.2.2 NRST External Reset Control

The Reset State Manager asserts the signal ext_nreset to assert the NRST pin. When this occurs, the “nrst_out” signal is driven low by the NRST Manager for a time programmed by the field ERSTL in RSTC_MR. This assertion duration, named EXTERNAL_RESET_LENGTH, lasts $2^{(ERSTL+1)}$ Slow Clock cycles. This gives the approximate duration of an assertion between 60 µs and 2 seconds. Note that ERSTL at 0 defines a two-cycle duration for the NRST pulse.

This feature allows the Reset Controller to shape the NRST pin level, and thus to guarantee that the NRST line is driven low for a time compliant with potential external devices connected on the system reset.

10.3.3 Reset States

The Reset State Manager handles the different reset sources and generates the internal reset signals. It reports the reset status in the field RSTTYP of the Status Register (RSTC_SR). The update of the field RSTTYP is performed when the processor reset is released.

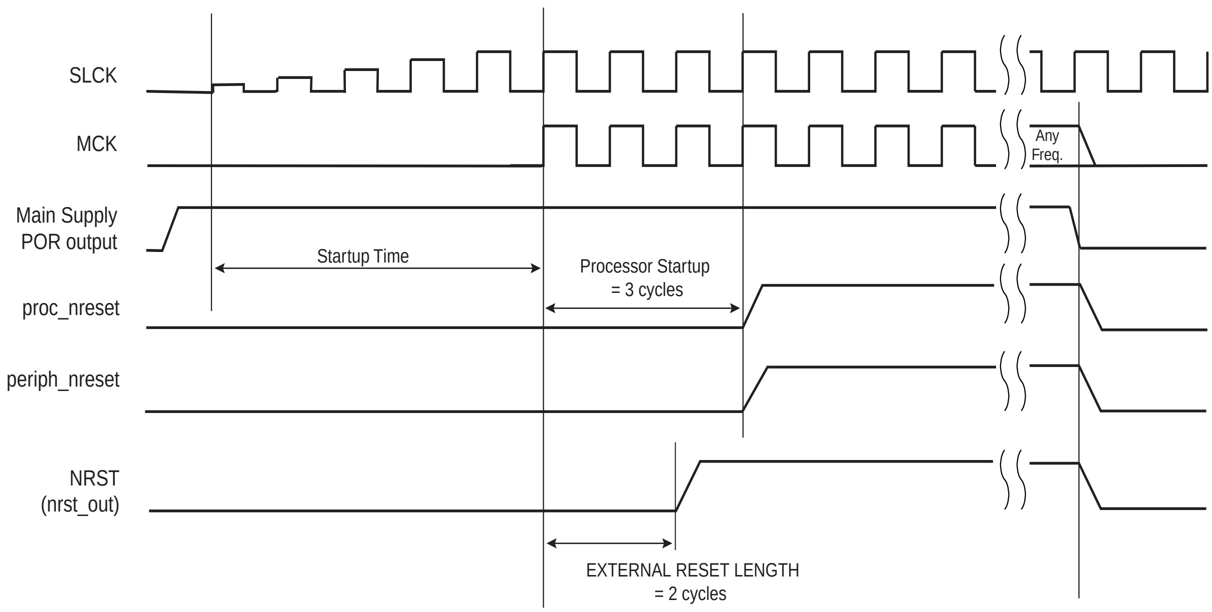
10.3.3.1 Power-up Reset

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

The startup time, as shown in Figure 10-3, is hardcoded to comply with the Slow Clock Oscillator startup time. After the startup time, the reset signals are released and the field RSTTYP in RSTC_SR reports a Power-up Reset.

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

Figure 10-3. Power-up Reset



10.3.3.2 User Reset

The User Reset is entered when a low level is detected on the NRST pin and the bit URSTEN in RSTC_MR is at 1. The NRST input signal is resynchronized with SLCK to insure proper behavior of the system.

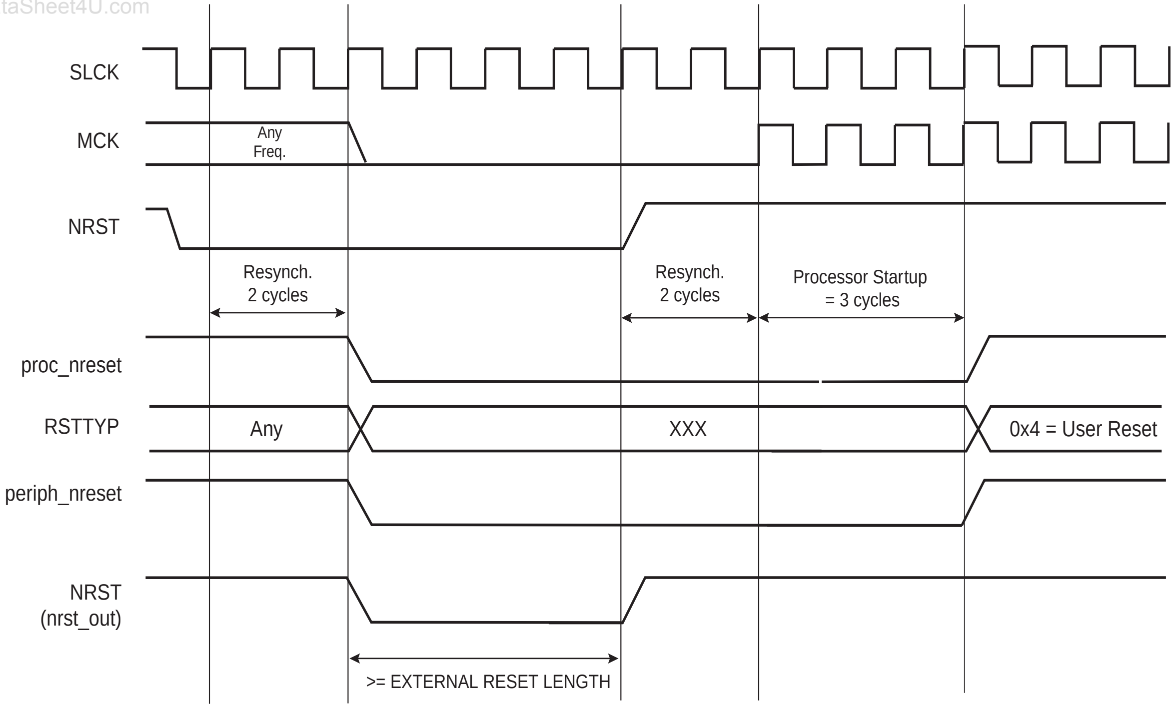
The User Reset is entered as soon as a low level is detected on NRST. The Processor Reset and the Peripheral Reset are asserted.

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

When the processor reset signal is released, the RSTTYP field of the Status Register (RSTC_SR) is loaded with the value 0x4, indicating a User Reset.

The NRST Manager guarantees that the NRST line is asserted for EXTERNAL_RESET_LENGTH Slow Clock cycles, as programmed in the field ERSTL. However, if NRST does not rise after EXTERNAL_RESET_LENGTH because it is driven low externally, the internal reset lines remain asserted until NRST actually rises.

Figure 10-4. User Reset State



10.3.3.3 Software Reset

The Reset Controller offers several commands used to assert the different reset signals. These commands are performed by writing the Control Register (RSTC_CR) with the following bits at 1:

- PROCRST: Writing PROCRST at 1 resets the processor and the watchdog timer.
- PERRST: Writing PERRST at 1 resets all the embedded peripherals, including the memory system, and, in particular, the Remap Command. The Peripheral Reset is generally used for debug purposes.
- EXTRST: Writing EXTRST at 1 asserts low the NRST pin during a time defined by the field ERSTL in the Mode Register (RSTC_MR).

The software reset is entered if at least one of these bits is set by the software. All these commands can be performed independently or simultaneously. The software reset lasts 2 Slow Clock cycles.

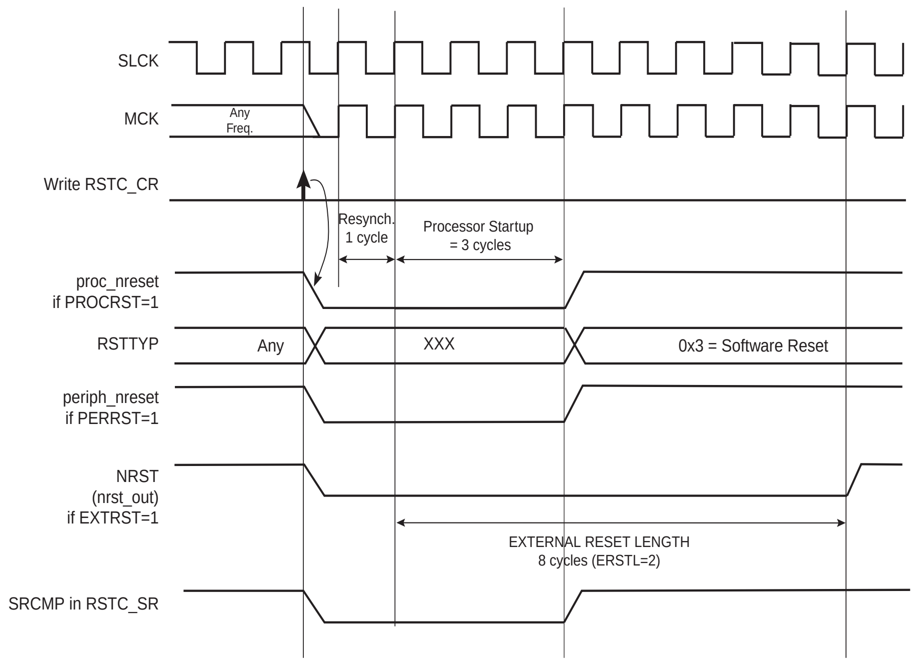
The internal reset signals are asserted as soon as the register write is performed. This is detected on the Master Clock (MCK). They are released when the software reset is left, i.e.; synchronously to SLCK.

If EXTRST is set, the nrst_out signal is asserted depending on the programming of the field ERSTL. However, the resulting falling edge on NRST does not lead to a User Reset.

If and only if the PROCRST bit is set, the Reset Controller reports the software status in the field RSTTYP of the Status Register (RSTC_SR). Other Software Resets are not reported in RSTTYP.

As soon as a software operation is detected, the bit SRCMP (Software Reset Command in Progress) is set in the Status Register (RSTC_SR). It is cleared as soon as the software reset is left. No other software reset can be performed while the SRCMP bit is set, and writing any value in RSTC_CR has no effect.

Figure 10-5. Software Reset



10.3.3.4 Watchdog Reset

The Watchdog Reset is entered when a watchdog fault occurs. This state lasts 2 Slow Clock cycles.

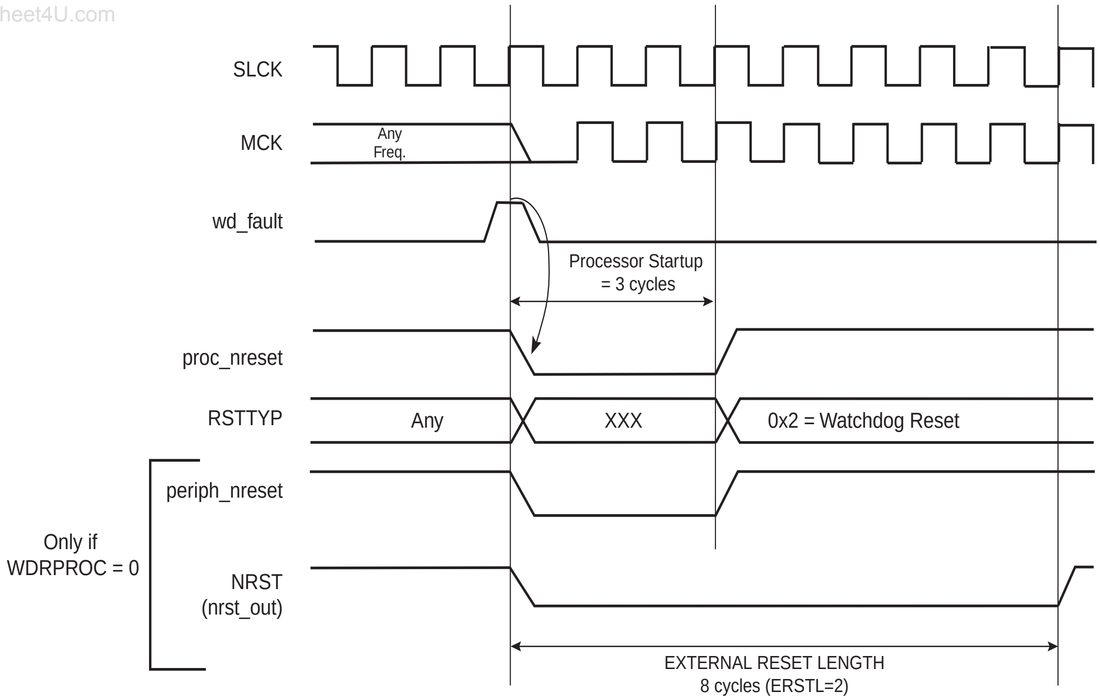
When in Watchdog Reset, assertion of the reset signals depends on the WDRPROC bit in WDT_MR:

- If WDRPROC is 0, the Processor Reset and the Peripheral Reset are asserted. The NRST line is also asserted, depending on the programming of the field ERSTL. However, the resulting low level on NRST does not result in a User Reset state.
- If WDRPROC = 1, only the processor reset is asserted.

The Watchdog Timer is reset by the proc_nreset signal. As the watchdog fault always causes a processor reset if WDRSTEN is set, the Watchdog Timer is always reset after a Watchdog Reset, and the Watchdog is enabled by default and with a period set to a maximum.

When the WDRSTEN in WDT_MR bit is reset, the watchdog fault has no impact on the reset controller.

Figure 10-6. Watchdog Reset



10.3.4 Reset State Priorities

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

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

Particular cases are listed below:

- When in User Reset:
  - A watchdog event is impossible because the Watchdog Timer is being reset by the proc_nreset signal.
  - A software reset is impossible, since the processor reset is being activated.
- When in Software Reset:
  - A watchdog event has priority over the current state.
  - The NRST has no effect.
- When in Watchdog Reset:
  - The processor reset is active and so a Software Reset cannot be programmed.
  - A User Reset cannot be entered.

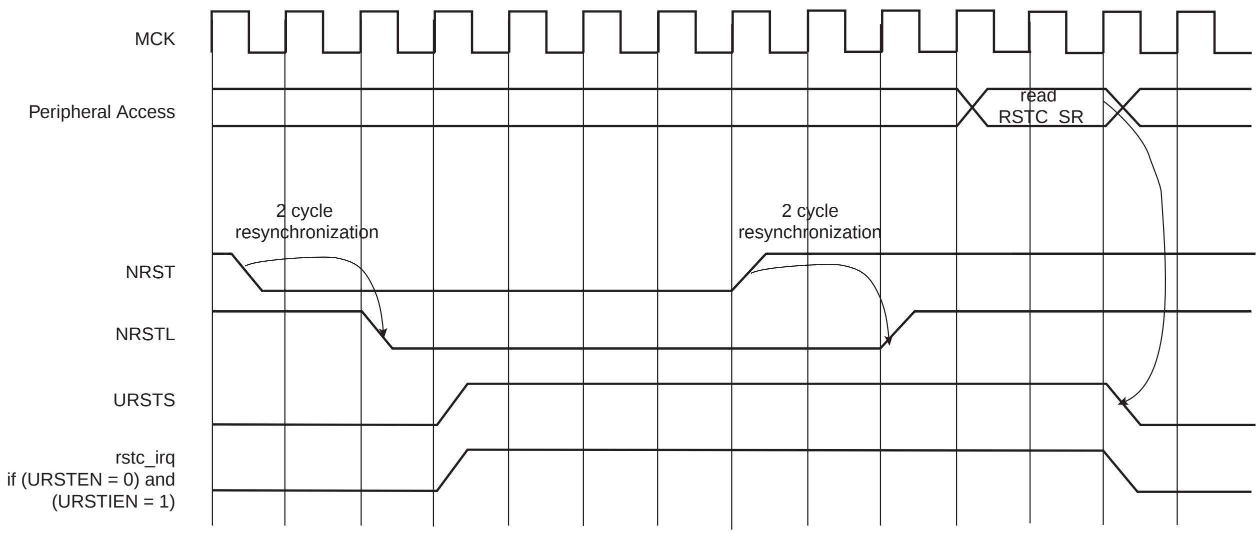
10.3.5 Reset Controller Status Register

The Reset Controller status register (RSTC_SR) provides several status fields:

- RSTTYP field: This field gives the type of the last reset, as explained in previous sections.

- SRCMP bit: This field indicates that a Software Reset Command is in progress and that no further software reset should be performed until the end of the current one. This bit is automatically cleared at the end of the current software reset.
- NRSTL bit: The NRSTL bit of the Status Register gives the level of the NRST pin sampled on each MCK rising edge.
- URSTS bit: A high-to-low transition of the NRST pin sets the URSTS bit of the RSTC_SR register. This transition is also detected on the Master Clock (MCK) rising edge (see Figure 10-7). If the User Reset is disabled (URSTEN = 0) and if the interruption is enabled by the URSTIEN bit in the RSTC_MR register, the URSTS bit triggers an interrupt. Reading the RSTC_SR status register resets the URSTS bit and clears the interrupt.

Figure 10-7. Reset Controller Status and Interrupt



10.4 Reset Controller (RSTC) User Interface

Table 10-1. Reset Controller (RSTC) Register Mapping

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

10.4.1 Reset Controller Control Register

Register Name: RSTC_CR

Access Type: Write-only

31	30	29	28	27	26	25	24
KEY							

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–		–

7	6	5	4	3	2	1	0
–	–	–	–	EXTRST	PERRST	–	PROCRST

- **PROCRST: Processor Reset**

0 = No effect.

1 = If KEY is correct, resets the processor.

- **PERRST: Peripheral Reset**

0 = No effect.

1 = If KEY is correct, resets the peripherals.

- **EXTRST: External Reset**

0 = No effect.

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

- **KEY: Password**

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

10.4.2 Reset Controller Status Register

Register Name: RSTC_SR

Access Type: Read-only

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

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

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

7	6	5	4	3	2	1	0
–	–	–	–	–	–		URSTS

- **URSTS: User Reset Status**

0 = No high-to-low edge on NRST happened since the last read of RSTC_SR.

1 = At least one high-to-low transition of NRST has been detected since the last read of RSTC_SR.

- **RSTTYP: Reset Type**

Reports the cause of the last processor reset. Reading this RSTC_SR does not reset this field.

Table 1.

RSTTYP			Reset Type	Comments
0	0	0	Power-up Reset	VDDCORE rising
0	1	0	Watchdog Reset	Watchdog fault occurred
0	1	1	Software Reset	Processor reset required by the software
1	0	0	User Reset	NRST pin detected low

- **NRSTL: NRST Pin Level**

Registers the NRST Pin Level at Master Clock (MCK).

- **SRCMP: Software Reset Command in Progress**

0 = No software command is being performed by the reset controller. The reset controller is ready for a software command.

1 = A software reset command is being performed by the reset controller. The reset controller is busy.

10.4.3 Reset Controller Mode Register

Register Name: RSTC_MR

Access Type: Read/Write

31	30	29	28	27	26	25	24
KEY							

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

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

7	6	5	4	3	2	1	0
–	–		URSTIEN	–	–	–	URSTEN

- **URSTEN: User Reset Enable**

0 = The detection of a low level on the pin NRST does not generate a User Reset.

1 = The detection of a low level on the pin NRST triggers a User Reset.

- **URSTIEN: User Reset Interrupt Enable**

0 = USRTS bit in RSTC_SR at 1 has no effect on rstc_irq.

1 = USRTS bit in RSTC_SR at 1 asserts rstc_irq if URSTEN = 0.

- **ERSTL: External Reset Length**

This field defines the external reset length. The external reset is asserted during a time of $2^{(ERSTL+1)}$ Slow Clock cycles. This allows assertion duration to be programmed between 60 µs and 2 seconds.

- **KEY: Password**

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

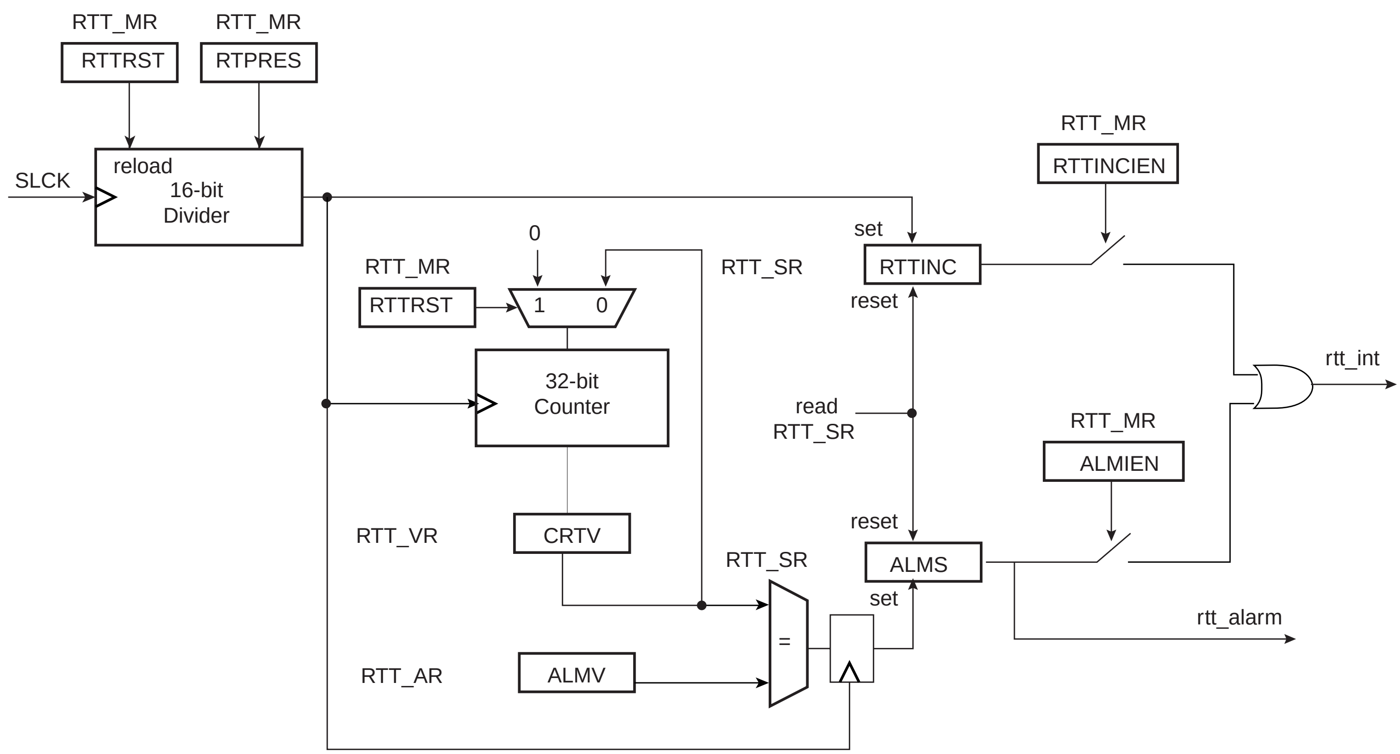
11. Real-time Timer (RTT)

11.1 Overview

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

11.2 Block Diagram

Figure 11-1. Real-time Timer



11.3 Functional Description

The Real-time Timer is used to count elapsed seconds. It is built around a 32-bit counter fed by Slow Clock divided by a programmable 16-bit value. The value can be programmed in the field RTPRES of the Real-time Mode Register (RTT_MR).

Programming RTPRES at 0x00008000 corresponds to feeding the real-time counter with a 1 Hz signal (if the Slow Clock is 32.768 Hz). The 32-bit counter can count up to 2^{32} seconds, corresponding to more than 136 years, then roll over to 0.

The Real-time Timer can also be used as a free-running timer with a lower time-base. The best accuracy is achieved by writing RTPRES to 3. Programming RTPRES to 1 or 2 is possible, but may result in losing status events because the status register is cleared two Slow Clock cycles after read. Thus if the RTT is configured to trigger an interrupt, the interrupt occurs during 2 Slow Clock cycles after reading RTT_SR. To prevent several executions of the interrupt handler, the interrupt must be disabled in the interrupt handler and re-enabled when the status register is clear.

The Real-time Timer value (CRTV) can be read at any time in the register RTT_VR (Real-time Value Register). As this value can be updated asynchronously from the Master Clock, it is advisable to read this register twice at the same value to improve accuracy of the returned value.

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

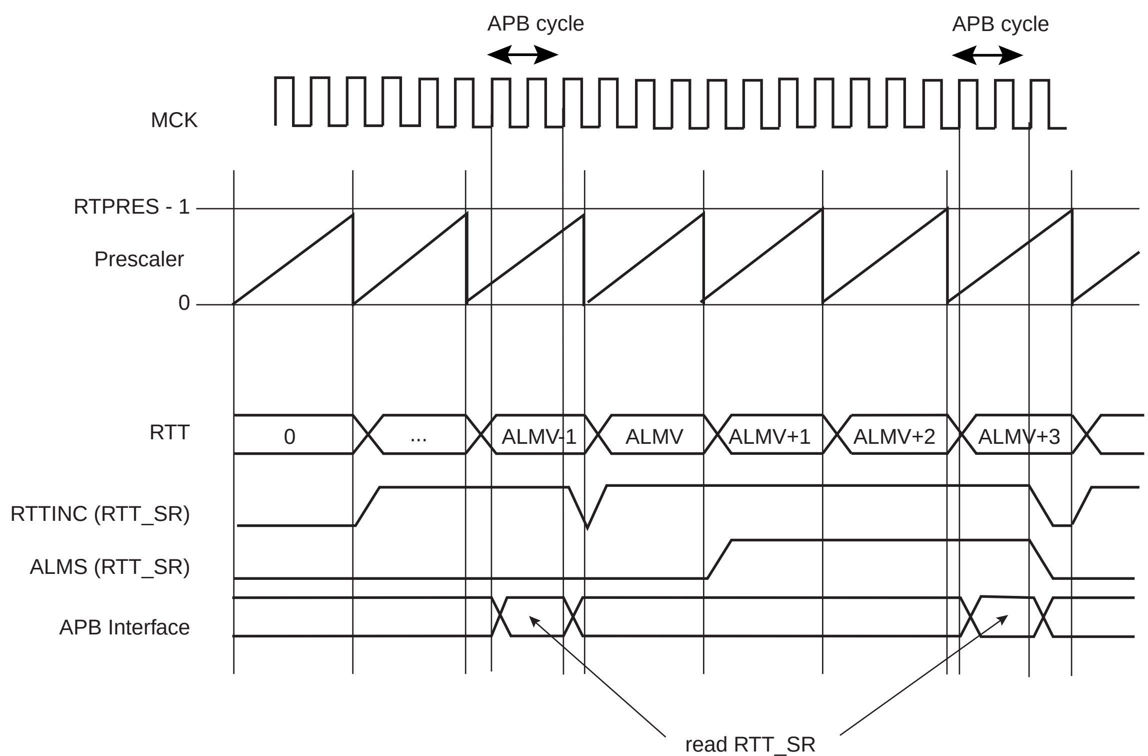
The bit RTTINC in RTT_SR is set each time the Real-time Timer counter is incremented. This bit can be used to start a periodic interrupt, the period being one second when the RTPRES is programmed with 0x8000 and Slow Clock equal to 32.768 Hz.

Reading the RTT_SR status register resets the RTTINC and ALMS fields.

Writing the bit RTTRST in RTT_MR immediately reloads and restarts the clock divider with the new programmed value. This also resets the 32-bit counter.

Note: Because of the asynchronism between the Slow Clock (SCLK) and the System Clock (MCK):
1) The restart of the counter and the reset of the RTT_VR current value register is effective only 2 slow clock cycles after the write of the RTTRST bit in the RTT_MR register.
2) The status register flags reset is taken into account only 2 slow clock cycles after the read of the RTT_SR (Status Register).

Figure 11-2. RTT Counting



11.4 Real-time Timer (RTT) User Interface

11.4.1 Register Mapping

Table 11-1. Real-time Timer Register Mapping

Offset	Register	Name	Access	Reset Value
0x20	Mode Register	RTT_MR	Read/Write	0x0000_8000
0x24	Alarm Register	RTT_AR	Read/Write	0xFFFF_FFFF
0x28	Value Register	RTT_VR	Read-only	0x0000_0000
0x2C	Status Register	RTT_SR	Read-only	0x0000_0000

11.4.2 Real-time Timer Mode Register

Register Name: RTT_MR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
–	–	–	–	–	RTTRST	RTTINCIEN	ALMIEN

15	14	13	12	11	10	9	8
RTPRES							

7	6	5	4	3	2	1	0
RTPRES							

- **RTPRES: Real-time Timer Prescaler Value**

Defines the number of SLCK periods required to increment the Real-time timer. RTPRES is defined as follows:

RTPRES = 0: The prescaler period is equal to 2^{16}

RTPRES ≠ 0: The period is equal to RTPRES.

- **ALMIEN: Alarm Interrupt Enable**

0 = The bit ALMS in RTT_SR has no effect on interrupt.

1 = The bit ALMS in RTT_SR asserts interrupt.

- **RTTINCIEN: Real-time Timer Increment Interrupt Enable**

0 = The bit RTTINC in RTT_SR has no effect on interrupt.

1 = The bit RTTINC in RTT_SR asserts interrupt.

- **RTTRST: Real-time Timer Restart**

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

11.4.3 Real-time Timer Alarm Register

Register Name: RTT_AR

Access Type: Read/Write

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

- **ALMV: Alarm Value**

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

11.4.4 Real-time Timer Value Register

Register Name: RTT_VR

Access Type: Read-only

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

- **CRTV: Current Real-time Value**

Returns the current value of the Real-time Timer.

11.4.5 Real-time Timer Status Register

Register Name: RTT_SR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	RTTINC	ALMS

- **ALMS: Real-time Alarm Status**

0 = The Real-time Alarm has not occured since the last read of RTT_SR.

1 = The Real-time Alarm occured since the last read of RTT_SR.

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

0 = The Real-time Timer has not been incremented since the last read of the RTT_SR.

1 = The Real-time Timer has been incremented since the last read of the RTT_SR.

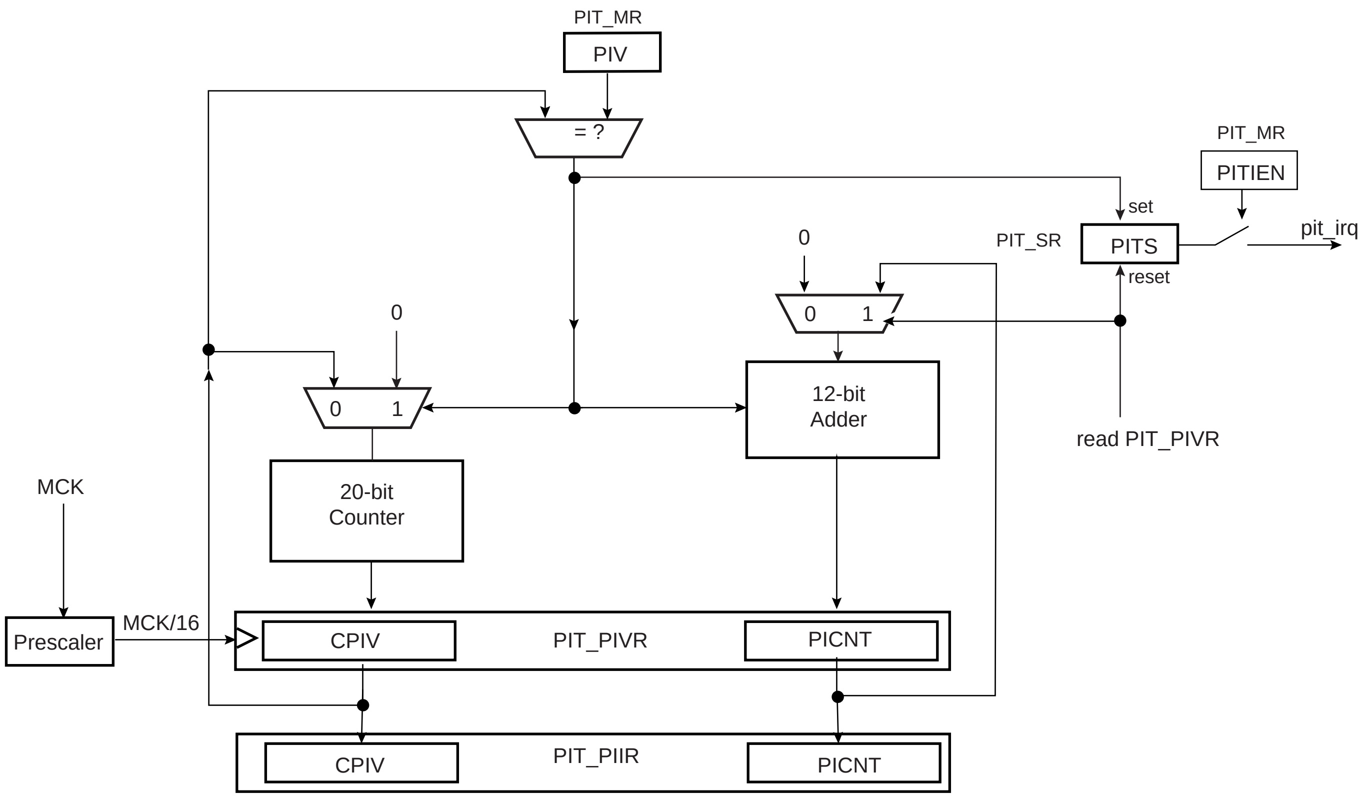
12. Periodic Interval Timer (PIT)

12.1 Overview

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

12.2 Block Diagram

Figure 12-1. Periodic Interval Timer



12.3 Functional Description

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

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

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

Writing a new PIV value in PIT_MR does not reset/restart the counters.

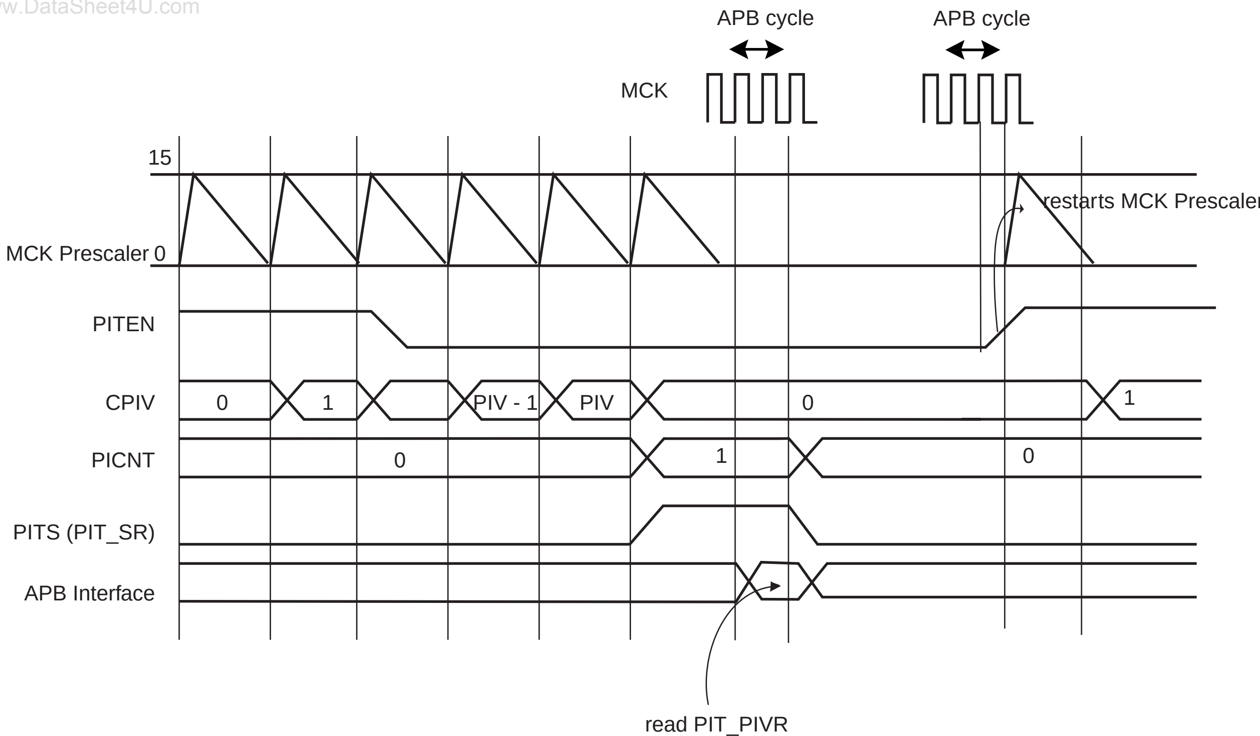
When CPIV and PICNT values are obtained by reading the Periodic Interval Value Register (PIT_PIVR), the overflow counter (PICNT) is reset and the PITS is cleared, thus acknowledging the interrupt. The value of PICNT gives the number of periodic intervals elapsed since the last read of PIT_PIVR.

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

The PIT may be enabled/disabled using the PITEN bit in the PIT_MR register (disabled on reset). The PITEN bit only becomes effective when the CPIV value is 0. Figure 12-2 illustrates the PIT counting. After the PIT Enable bit is reset (PITEN= 0), the CPIV goes on counting until the PIV value is reached, and is then reset. PIT restarts counting, only if the PITEN is set again.

The PIT is stopped when the core enters debug state.

Figure 12-2. Enabling/Disabling PIT with PITEN



12.4 Periodic Interval Timer (PIT) User Interface

Table 12-1. Periodic Interval Timer (PIT) Register Mapping

Offset	Register	Name	Access	Reset Value
0x30	Mode Register	PIT_MR	Read/Write	0x000F_FFFF
0x34	Status Register	PIT_SR	Read-only	0x0000_0000
0x38	Periodic Interval Value Register	PIT_PIVR	Read-only	0x0000_0000
0x3C	Periodic Interval Image Register	PIT_PIIR	Read-only	0x0000_0000

12.4.1 Periodic Interval Timer Mode Register

Register Name: PIT_MR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
PIV							

7	6	5	4	3	2	1	0
PIV							

- **PIV: Periodic Interval Value**

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

- **PITEN: Period Interval Timer Enabled**

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

1 = The Periodic Interval Timer is enabled.

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

0 = The bit PITS in PIT_SR has no effect on interrupt.

1 = The bit PITS in PIT_SR asserts interrupt.

12.4.2 Periodic Interval Timer Status Register

Register Name: PIT_SR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	PITS

- **PITS: Periodic Interval Timer Status**

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

1 = The Periodic Interval timer has reached PIV since the last read of PIT_PIVR.

12.4.3 Periodic Interval Timer Value Register

Register Name: PIT_PIVR

Access Type: Read-only

31	30	29	28	27	26	25	24
PICNT							

23	22	21	20	19	18	17	16
PICNT				CPIV			

15	14	13	12	11	10	9	8
CPIV							

7	6	5	4	3	2	1	0
CPIV							

Reading this register clears PITS in PIT_SR.

- **CPIV: Current Periodic Interval Value**

Returns the current value of the periodic interval timer.

- **PICNT: Periodic Interval Counter**

Returns the number of occurrences of periodic intervals since the last read of PIT_PIVR.

12.4.4 Periodic Interval Timer Image Register

Register Name: PIT_PIIR

Access Type: Read-only

31	30	29	28	27	26	25	24
PICNT							

23	22	21	20	19	18	17	16
PICNT				CPIV			

15	14	13	12	11	10	9	8
CPIV							

7	6	5	4	3	2	1	0
CPIV							

- **CPIV: Current Periodic Interval Value**

Returns the current value of the periodic interval timer.

- **PICNT: Periodic Interval Counter**

Returns the number of occurrences of periodic intervals since the last read of PIT_PIVR.

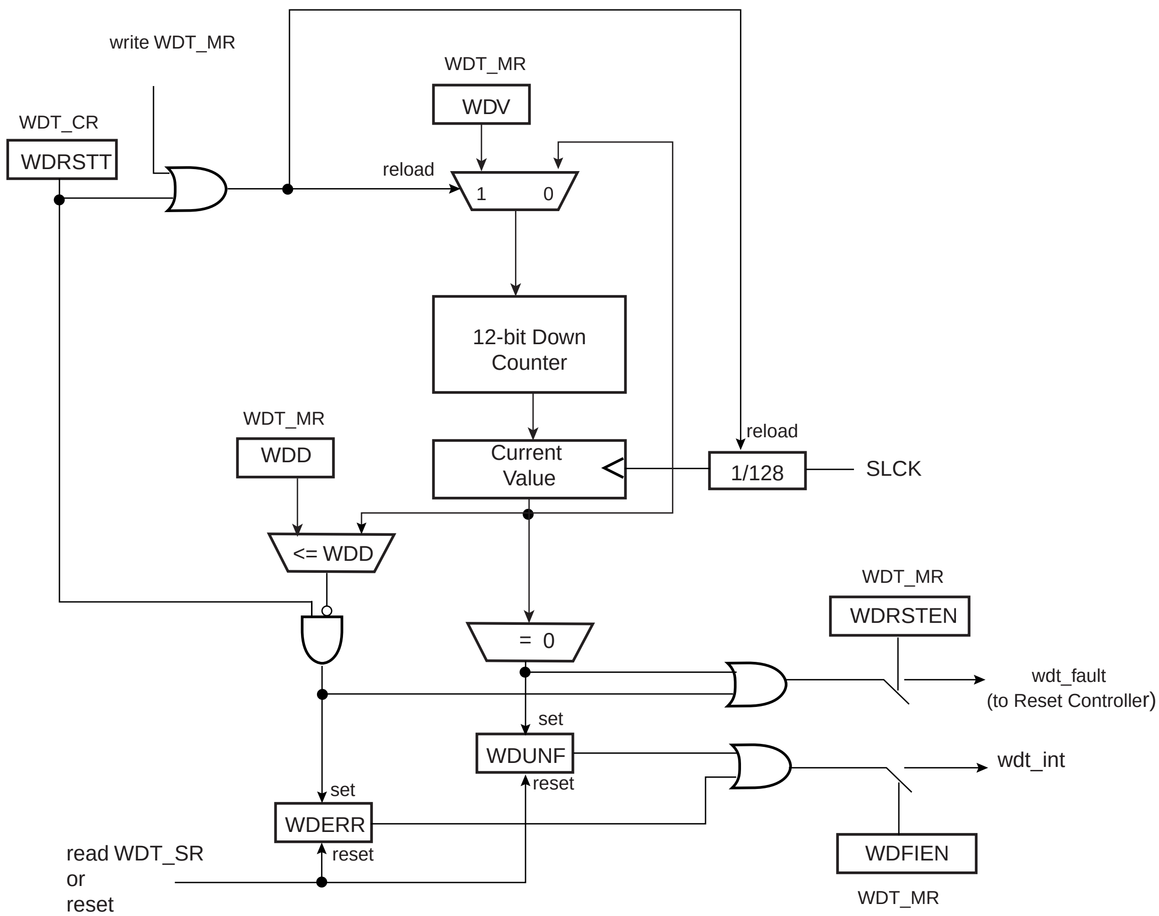
13. Watchdog Timer (WDT)

13.1 Overview

The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It features a 12-bit down counter that allows a watchdog period of up to 16 seconds (slow clock at 32.768 kHz). It can generate a general reset or a processor reset only. In addition, it can be stopped while the processor is in debug mode or idle mode.

13.2 Block Diagram

Figure 13-1. Watchdog Timer Block Diagram



13.3 Functional Description

The Watchdog Timer can be used to prevent system lock-up if the software becomes trapped in a deadlock. It is supplied with VDDCORE. It restarts with initial values on processor reset.

The Watchdog is built around a 12-bit down counter, which is loaded with the value defined in the field WV of the Mode Register (WDT_MR). The Watchdog Timer uses the Slow Clock divided by 128 to establish the maximum Watchdog period to be 16 seconds (with a typical Slow Clock of 32.768 kHz).

After a Processor Reset, the value of WV is 0xFFF, corresponding to the maximum value of the counter with the external reset generation enabled (field WDRSTEN at 1 after a Backup Reset). This means that a default Watchdog is running at reset, i.e., at power-up. The user must either disable it (by setting the WDDIS bit in WDT_MR) if he does not expect to use it or must reprogram it to meet the maximum Watchdog period the application requires.

The Watchdog Mode Register (WDT_MR) can be written only once. Only a processor reset resets it. Writing the WDT_MR register reloads the timer with the newly programmed mode parameters.

In normal operation, the user reloads the Watchdog at regular intervals before the timer underflow occurs, by writing the Control Register (WDT_CR) with the bit WDRSTT to 1. The Watchdog counter is then immediately reloaded from WDT_MR and restarted, and the Slow Clock 128 divider is reset and restarted. The WDT_CR register is write-protected. As a result, writing WDT_CR without the correct hard-coded key has no effect. If an underflow does occur, the “wdt_fault” signal to the Reset Controller is asserted if the bit WDRSTEN is set in the Mode Register (WDT_MR). Moreover, the bit WDUNF is set in the Watchdog Status Register (WDT_SR).

To prevent a software deadlock that continuously triggers the Watchdog, the reload of the Watchdog must occur while the Watchdog counter is within a window between 0 and WDD, WDD is defined in the WatchDog Mode Register WDT_MR.

Any attempt to restart the Watchdog while the Watchdog counter is between WDV and WDD results in a Watchdog error, even if the Watchdog is disabled. The bit WDERR is updated in the WDT_SR and the “wdt_fault” signal to the Reset Controller is asserted.

Note that this feature can be disabled by programming a WDD value greater than or equal to the WDV value. In such a configuration, restarting the Watchdog Timer is permitted in the whole range [0; WDV] and does not generate an error. This is the default configuration on reset (the WDD and WDV values are equal).

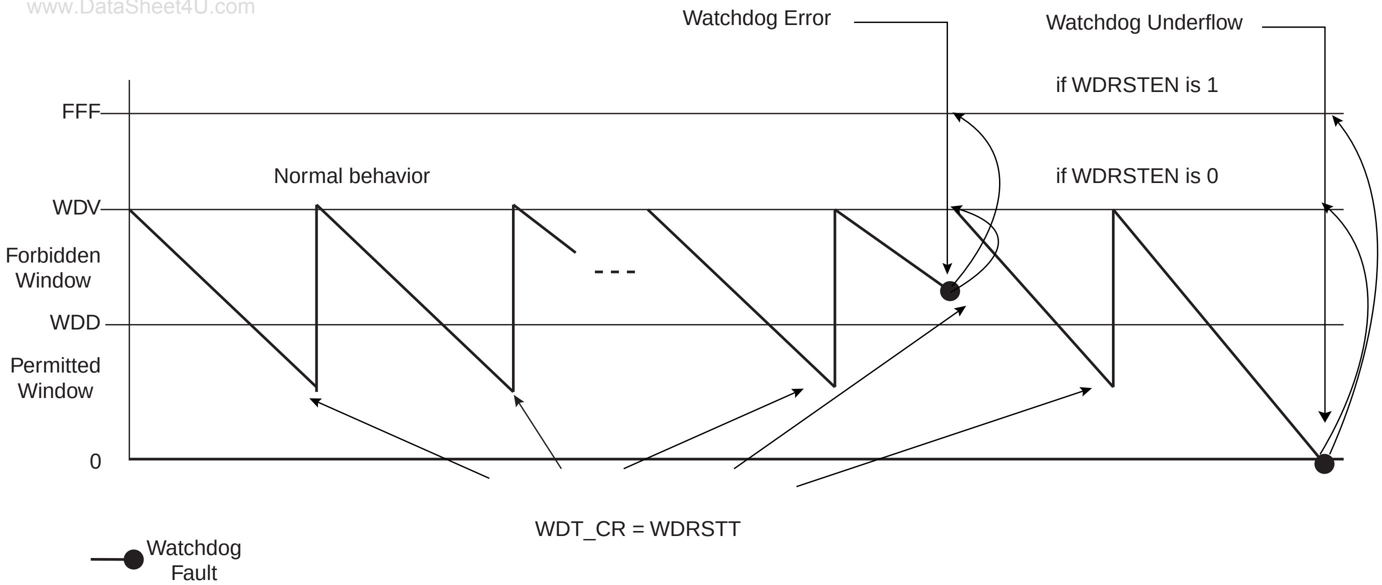
The status bits WDUNF (Watchdog Underflow) and WDERR (Watchdog Error) trigger an interrupt, provided the bit WDFIEN is set in the mode register. The signal “wdt_fault” to the reset controller causes a Watchdog reset if the WDRSTEN bit is set as already explained in the reset controller programmer Datasheet. In that case, the processor and the Watchdog Timer are reset, and the WDERR and WDUNF flags are reset.

If a reset is generated or if WDT_SR is read, the status bits are reset, the interrupt is cleared, and the “wdt_fault” signal to the reset controller is deasserted.

Writing the WDT_MR reloads and restarts the down counter.

While the processor is in debug state or in idle mode, the counter may be stopped depending on the value programmed for the bits WDIDLEHLT and WDDBGHLT in the WDT_MR.

Figure 13-2. Watchdog Behavior



13.4 Watchdog Timer (WDT) User Interface

Table 13-1. Watchdog Timer (WDT) Register Mapping

Offset	Register	Name	Access	Reset Value
0x40	Control Register	WDT_CR	Write-only	-
0x44	Mode Register	WDT_MR	Read/Write Once	0x3FFF_2FFF
0x48	Status Register	WDT_SR	Read-only	0x0000_0000

13.4.1 Watchdog Timer Control Register

Register Name: WDT_CR

Access Type: Write-only

31	30	29	28	27	26	25	24
KEY							

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	WDRSTT

- **WDRSTT: Watchdog Restart**

0: No effect.

1: Restarts the Watchdog.

- **KEY: Password**

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

13.4.2 Watchdog Timer Mode Register

Register Name: WDT_MR

Access Type: Read/Write Once

31	30	29	28	27	26	25	24
–	–	WDIDLEHLT	WDDBGHLT	WDD			

23	22	21	20	19	18	17	16
WDD							

15	14	13	12	11	10	9	8
WDDIS	WDRPROC	WDRSTEN	WDFIEN	WDV			

7	6	5	4	3	2	1	0
WDV							

- **WDV: Watchdog Counter Value**

Defines the value loaded in the 12-bit Watchdog Counter.

- **WDFIEN: Watchdog Fault Interrupt Enable**

0: A Watchdog fault (underflow or error) has no effect on interrupt.

1: A Watchdog fault (underflow or error) asserts interrupt.

- **WDRSTEN: Watchdog Reset Enable**

0: A Watchdog fault (underflow or error) has no effect on the resets.

1: A Watchdog fault (underflow or error) triggers a Watchdog reset.

- **WDRPROC: Watchdog Reset Processor**

0: If WDRSTEN is 1, a Watchdog fault (underflow or error) activates all resets.

1: If WDRSTEN is 1, a Watchdog fault (underflow or error) activates the processor reset.

- **WDD: Watchdog Delta Value**

Defines the permitted range for reloading the Watchdog Timer.

If the Watchdog Timer value is less than or equal to WDD, writing WDT_CR with WDRSTT = 1 restarts the timer.

If the Watchdog Timer value is greater than WDD, writing WDT_CR with WDRSTT = 1 causes a Watchdog error.

- **WDDBGHLT: Watchdog Debug Halt**

0: The Watchdog runs when the processor is in debug state.

1: The Watchdog stops when the processor is in debug state.

- **WDIDLEHLT: Watchdog Idle Halt**

0: The Watchdog runs when the system is in idle mode.

1: The Watchdog stops when the system is in idle state.

- **WDDIS: Watchdog Disable**

0: Enables the Watchdog Timer.

1: Disables the Watchdog Timer.

13.4.3 Watchdog Timer Status Register

Register Name: WDT_SR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	WDERR	WDUNF

- **WDUNF: Watchdog Underflow**

0: No Watchdog underflow occurred since the last read of WDT_SR.

1: At least one Watchdog underflow occurred since the last read of WDT_SR.

- **WDERR: Watchdog Error**

0: No Watchdog error occurred since the last read of WDT_SR.

1: At least one Watchdog error occurred since the last read of WDT_SR.

14. Bus Matrix

14.1 Description

The Bus Matrix implements a multi-layer AHB, based on the AHB-Lite protocol, that enables parallel access paths between multiple AHB masters and slaves in a system, thus increasing the overall bandwidth. The Bus Matrix interconnects up to 16 AHB Masters to up to 16 AHB Slaves. The normal latency to connect a master to a slave is one cycle except for the default master of the accessed slave which is connected directly (zero cycle latency). The Bus Matrix user interface is compliant with ARM Advance Peripheral Bus and provides 16 Special Function Registers (MATRIX_SFR) that allow the Bus Matrix to support application specific features.

14.2 Memory Mapping

The Bus Matrix provides one decoder for every AHB Master Interface. The decoder offers each AHB Master several memory mappings. In fact, depending on the product, each memory area may be assigned to several slaves. Booting at the same address while using different AHB slaves (i.e. external RAM, internal ROM or internal Flash, etc.) becomes possible.

The Bus Matrix user interface provides Master Remap Control Register (MATRIX_MRCR) that performs remap action for every master independently.

14.3 Special Bus Granting Mechanism

The Bus Matrix provides some speculative bus granting techniques in order to anticipate access requests from some masters. This mechanism reduces latency at first access of a burst or single transfer. This bus granting mechanism sets a different default master for every slave.

At the end of the current access, if no other request is pending, the slave remains connected to its associated default master. A slave can be associated with three kinds of default masters: no default master, last access master and fixed default master.

14.3.1 No Default Master

At the end of the current access, if no other request is pending, the slave is disconnected from all masters. No Default Master suits low-power mode.

14.3.2 Last Access Master

At the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

14.3.3 Fixed Default Master

At the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike last access master, the fixed master does not change unless the user modifies it by a software action (field FIXED_DEFMSTR of the related MATRIX_SCFG).

To change from one kind of default master to another, the Bus Matrix user interface provides the Slave Configuration Registers, one for each slave, that set a default master for each slave. The Slave Configuration Register contains two fields: DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field selects the default master type (no default, last access master, fixed default master), whereas the 4-bit FIXED_DEFMSTR field selects a fixed default master provided that DEFMSTR_TYPE is set to fixed default master. Please refer to the Bus Matrix user interface description.

14.4 Arbitration

The Bus Matrix provides an arbitration mechanism that reduces latency when conflict cases occur, i.e. when two or more masters try to access the same slave at the same time. One arbiter per AHB slave is provided, thus arbitrating each slave differently.

The Bus Matrix provides the user with the possibility of choosing between 2 arbitration types for each slave:

1. Round-Robin Arbitration (default)
2. Fixed Priority Arbitration

This choice is made via the field ARBT of the Slave Configuration Registers (MATRIX_SCFG).

Each algorithm may be complemented by selecting a default master configuration for each slave.

When a re-arbitration must be done, specific conditions apply. See Section 14.4.1 "Arbitration Rules" on page 121.

14.4.1 Arbitration Rules

Each arbiter has the ability to arbitrate between two or more different master requests. In order to avoid burst breaking and also to provide the maximum throughput for slave interfaces, arbitration may only take place during the following cycles:

1. Idle Cycles: When a slave is not connected to any master or is connected to a master which is not currently accessing it.
2. Single Cycles: When a slave is currently doing a single access.
3. End of Burst Cycles: When the current cycle is the last cycle of a burst transfer. For defined length burst, predicted end of burst matches the size of the transfer but is managed differently for undefined length burst. See "Undefined Length Burst Arbitration" on page 121.
4. Slot Cycle Limit: When the slot cycle counter has reached the limit value indicating that the current master access is too long and must be broken. See "Slot Cycle Limit Arbitration" on page 122.

14.4.1.1 Undefined Length Burst Arbitration

In order to avoid long slave handling during undefined length bursts (INCR), the Bus Matrix provides specific logic in order to re-arbitrate before the end of the INCR transfer. A predicted end of burst is used as a defined length burst transfer and can be selected from among the following five possibilities:

1. Infinite: No predicted end of burst is generated and therefore INCR burst transfer will never be broken.
2. One beat bursts: Predicted end of burst is generated at each single transfer inside the INCP transfer.
3. Four beat bursts: Predicted end of burst is generated at the end of each four beat boundary inside INCR transfer.
4. Eight beat bursts: Predicted end of burst is generated at the end of each eight beat boundary inside INCR transfer.
5. Sixteen beat bursts: Predicted end of burst is generated at the end of each sixteen beat boundary inside INCR transfer.

This selection can be done through the field ULBT of the Master Configuration Registers (MATRIX_MCFG).

14.4.1.2 Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break long accesses, such as very long bursts on a very slow slave (e.g., an external low speed memory). At the beginning of the burst access, a counter is loaded with the value previously written in the SLOT_CYCLE field of the related Slave Configuration Register (MATRIX_SCFG) and decreased at each clock cycle. When the counter reaches zero, the arbiter has the ability to re-arbitrate at the end of the current byte, half word or word transfer.

14.4.2 Round-Robin Arbitration

This algorithm allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave in a round-robin manner. If two or more master requests arise at the same time, the master with the lowest number is first serviced, then the others are serviced in a round-robin manner.

There are three round-robin algorithms implemented:

- Round-Robin arbitration without default master
- Round-Robin arbitration with last default master
- Round-Robin arbitration with fixed default master

14.4.2.1 Round-Robin Arbitration without Default Master

This is the main algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to dispatch requests from different masters to the same slave in a pure round-robin manner. At the end of the current access, if no other request is pending, the slave is disconnected from all masters. This configuration incurs one latency cycle for the first access of a burst. Arbitration without default master can be used for masters that perform significant bursts.

14.4.2.2 Round-Robin Arbitration with Last Default Master

This is a biased round-robin algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to remove the one latency cycle for the last master that accessed the slave. In fact, at the end of the current transfer, if no other master request is pending, the slave remains connected to the last master that performed the access. Other non privileged masters still get one latency cycle if they want to access the same slave. This technique can be used for masters that mainly perform single accesses.

14.4.2.3 Round-Robin Arbitration with Fixed Default Master

This is another biased round-robin algorithm. It allows the Bus Matrix arbiters to remove the one latency cycle for the fixed default master per slave. At the end of the current access, the slave remains connected to its fixed default master. Every request attempted by this fixed default master will not cause any latency whereas other non privileged masters will still get one latency cycle. This technique can be used for masters that mainly perform single accesses.

14.4.3 Fixed Priority Arbitration

This algorithm allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave by using the fixed priority defined by the user. If two or more master requests are active at the same time, the master with the highest priority number is serviced first. If two or more master requests with the same priority are active at the same time, the master with the highest number is serviced first.

For each slave, the priority of each master may be defined through the Priority Registers for Slaves (MATRIX_PRAS and MATRIX_PRBS).

14.5 AHB Generic Bus Matrix User Interface

Table 14-1. Register Mapping

Offset	Register	Name	Access	Reset Value
0x0000	Master Configuration Register 0	MATRIX_MCFG0	Read/Write	0x00000002
0x0004	Master Configuration Register 1	MATRIX_MCFG1	Read/Write	0x00000002
0x0008	Master Configuration Register 2	MATRIX_MCFG2	Read/Write	0x00000002
0x000C	Master Configuration Register 3	MATRIX_MCFG3	Read/Write	0x00000002
0x0010	Master Configuration Register 4	MATRIX_MCFG4	Read/Write	0x00000002
0x0014	Master Configuration Register 5	MATRIX_MCFG5	Read/Write	0x00000002
0x0018	Master Configuration Register 6	MATRIX_MCFG6	Read/Write	0x00000002
0x0040	Slave Configuration Register 0	MATRIX_SCFG0	Read/Write	0x00000010
0x0044	Slave Configuration Register 1	MATRIX_SCFG1	Read/Write	0x00000010
0x0048	Slave Configuration Register 2	MATRIX_SCFG2	Read/Write	0x00000010
0x004C	Slave Configuration Register 3	MATRIX_SCFG3	Read/Write	0x00000010
0x0050	Slave Configuration Register 4	MATRIX_SCFG4	Read/Write	0x00000010
0x0080	Priority Register A for Slave 0	MATRIX_PRAS0	Read/Write	0x00000000
0x0084	Priority Register B for Slave 0	MATRIX_PRBS0	Read/Write	0x00000000
0x0088	Priority Register A for Slave 1	MATRIX_PRAS1	Read/Write	0x00000000
0x008C	Priority Register B for Slave 1	MATRIX_PRBS1	Read/Write	0x00000000
0x0090	Priority Register A for Slave 2	MATRIX_PRAS2	Read/Write	0x00000000
0x0094	Priority Register B for Slave 2	MATRIX_PRBS2	Read/Write	0x00000000
0x0098	Priority Register A for Slave 3	MATRIX_PRAS3	Read/Write	0x00000000
0x009C	Priority Register B for Slave 3	MATRIX_PRBS3	Read/Write	0x00000000
0x00A0	Priority Register A for Slave 4	MATRIX_PRAS4	Read/Write	0x00000000
0x00A4	Priority Register B for Slave 4	MATRIX_PRBS4	Read/Write	0x00000000
0x0100	Master Remap Control Register	MATRIX_MRCR	Read/Write	0x00000000
0x0104 - 0x010C	Reserved	–	–	–
0x0110	Special Function Register 0	MATRIX_SFR0	Read/Write	0x00000000
0x0114	Special Function Register 1	MATRIX_SFR1	Read/Write	0x00000000
0x0118	Special Function Register 2	MATRIX_SFR2	Read/Write	0x00000000
0x011C	Special Function Register 3	MATRIX_SFR3	Read/Write	0x00000000
0x0120	Special Function Register 4	MATRIX_SFR4	Read/Write	0x00000000
0x0124	Special Function Register 5	MATRIX_SFR5	Read/Write	0x00000000
0x0128	Special Function Register 6	MATRIX_SFR6	Read/Write	0x00000000
0x012C	Special Function Register 7	MATRIX_SFR7	Read/Write	0x00000000
0x0130	Special Function Register 8	MATRIX_SFR8	Read/Write	0x00000000
0x0134	Special Function Register 9	MATRIX_SFR9	Read/Write	0x00000000
0x0138	Special Function Register 10	MATRIX_SFR10	Read/Write	0x00000000

Table 14-1. Register Mapping

Offset	Register	Name	Access	Reset Value
0x013C	Special Function Register 11	MATRIX_SFR11	Read/Write	0x00000000
0x0140	Special Function Register 12	MATRIX_SFR12	Read/Write	0x00000000
0x0144	Special Function Register 13	MATRIX_SFR13	Read/Write	0x00000000
0x0148	Special Function Register 14	MATRIX_SFR14	Read/Write	0x00000000
0x014C	Special Function Register 15	MATRIX_SFR15	Read/Write	0x00000000
0x0150 - 0x01F8	Reserved	–	–	–

14.5.1 Bus Matrix Master Configuration Registers

Register Name: MATRIX_MCFG0...MATRIX_MCFG15

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
–	–	–	–	–	ULBT		

- **ULBT: Undefined Length Burst Type**

0: Infinite Length Burst

No predicted end of burst is generated and therefore INCR bursts coming from this master cannot be broken.

1: Single Access

The undefined length burst is treated as a succession of single accesses, allowing re-arbitration at each beat of the INCR burst.

2: Four Beat Burst

The undefined length burst is split into a four-beat burst, allowing re-arbitration at each four-beat burst end.

3: Eight Beat Burst

The undefined length burst is split into an eight-beat burst, allowing re-arbitration at each eight-beat burst end.

4: Sixteen Beat Burst

The undefined length burst is split into a sixteen-beat burst, allowing re-arbitration at each sixteen-beat burst end.

14.5.2 Bus Matrix Slave Configuration Registers

Register Name: MATRIX_SCFG0...MATRIX_SCFG15

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
SLOT_CYCLE							

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

When the SLOT_CYCLE limit is reached for a burst, it may be broken by another master trying to access this slave.

This limit has been placed to avoid locking a very slow slave when very long bursts are used.

This limit must not be very small. Unreasonably small values break every burst and the Bus Matrix arbitrates without performing any data transfer. 16 cycles is a reasonable value for SLOT_CYCLE.

- **DEFMSTR_TYPE: Default Master Type**

0: No Default Master

At the end of the current slave access, if no other master request is pending, the slave is disconnected from all masters.

This results in a one cycle latency for the first access of a burst transfer or for a single access.

1: Last Default Master

At the end of the current slave access, if no other master request is pending, the slave stays connected to the last master having accessed it.

This results in not having one cycle latency when the last master tries to access the slave again.

2: Fixed Default Master

At the end of the current slave access, if no other master request is pending, the slave connects to the fixed master the number that has been written in the FIXED_DEFMSTR field.

This results in not having one cycle latency when the fixed master tries to access the slave again.

- **FIXED_DEFMSTR: Fixed Default Master**

This is the number of the Default Master for this slave. Only used if DEFMSTR_TYPE is 2. Specifying the number of a master which is not connected to the selected slave is equivalent to setting DEFMSTR_TYPE to 0.

- **ARBT: Arbitration Type**

0: Round-Robin Arbitration

1: Fixed Priority Arbitration

2: Reserved

3: Reserved

14.5.3 Bus Matrix Priority Registers A For Slaves

Register Name: MATRIX_PRAS0...MATRIX_PRAS15

Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	M7PR		–	–	M6PR	

23	22	21	20	19	18	17	16
–	–	M5PR		–	–	M4PR	

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

7	6	5	4	3	2	1	0
–	–	M1PR		–	–	M0PR	

- **MxPR: Master x Priority**

Fixed priority of Master x for accessing the selected slave. The higher the number, the higher the priority.

14.5.4 Bus Matrix Priority Registers B For Slaves

Register Name: MATRIX_PRBS0...MATRIX_PRBS15

Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	M15PR		–	–	M14PR	

23	22	21	20	19	18	17	16
–	–	M13PR		–	–	M12PR	

15	14	13	12	11	10	9	8
–	–	M11PR		–	–	M10PR	

7	6	5	4	3	2	1	0
–	–	M9PR		–	–	M8PR	

- **MxPR: Master x Priority**

Fixed priority of Master x for accessing the selected slave. The higher the number, the higher the priority.

14.5.5 Bus Matrix Master Remap Control Register

Register Name: MATRIX_MRCR

Access Type: Read/Write

Reset: 0x0000_0000

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

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

15	14	13	12	11	10	9	8
RCB15	RCB14	RCB13	RCB12	RCB11	RCB10	RCB9	RCB8

7	6	5	4	3	2	1	0
RCB7	RCB6	RCB5	RCB4	RCB3	RCB2	RCB1	RCB0

- **RCB: Remap Command Bit for Master x**

0: Disable remapped address decoding for the selected Master

1: Enable remapped address decoding for the selected Master

14.5.6 Bus Matrix Special Function Registers

Register Name: MATRIX_SFR0...MATRIX_SFR15

Access Type: Read/Write

Reset: 0x0000_0000

31	30	29	28	27	26	25	24
SFR							

23	22	21	20	19	18	17	16
SFR							

15	14	13	12	11	10	9	8
SFR							

7	6	5	4	3	2	1	0
SFR							

- **SFR: Special Function Register Fields**

The SFR fields are a set of D-type Flip-flops which are only connected to outputs of the Bus Matrix.

They are readable/writable from the User Interface and may be used to implement Configuration Registers which cannot be implemented in any of the other embedded peripherals of the product. Each bit of the SFR may be removed by hardware customization at synthesis if not used.

The only meaningful SFR are:

SFR0 = SFR_HTCM: bits 7-4 = DRSIZE; bits 3-0 = IRSIZE

0000 = Data (Instruction) TCM size = 0 KB

0101 = Data (Instruction) TCM size = 16 KB

0110 = Data (Instruction) TCM size = 32 KB

and

SFR3 = SFR_HEBI:

bit 1 = NCS1 selects SDRAM (1) or generic static memory (0)

bit 3 = NCS3 selects Smart Media (1) or generic static memory (0)

bit 4 = NCS4 selects Compact Flash slot0 (1) or generic static memory (0)

bit 5 = NCS5 selects Compact Flash slot1 (1) or generic static memory (0)

bit 8 = pullup applied on EBI data (1) or no pullup (0)

15. External Bus Interface (EBI)

15.1 Overview

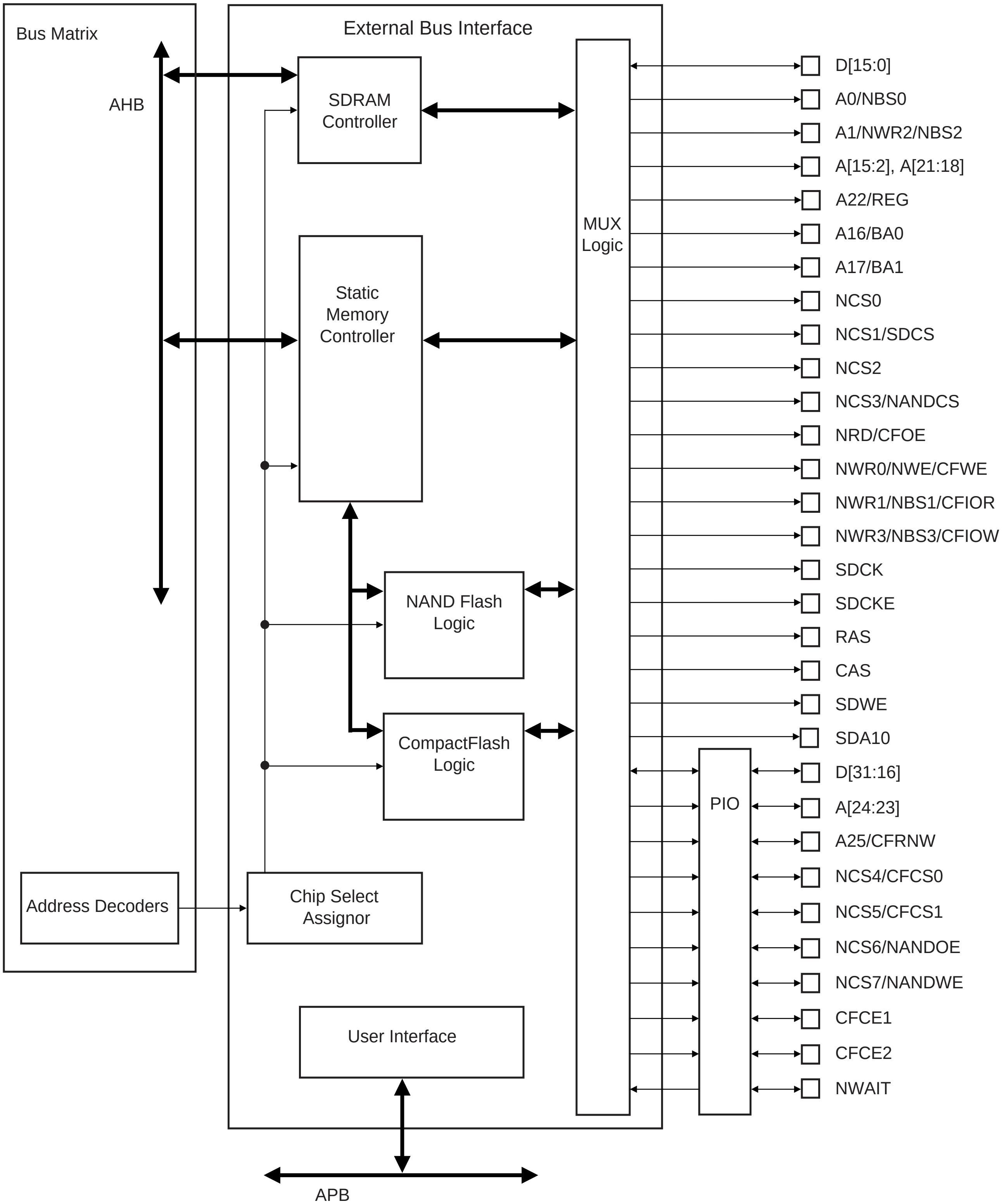
The External Bus Interface (EBI) is designed to ensure the successful data transfer between several external devices and the embedded Memory Controller of an ARM-based device. The Static Memory and SDRAM Controllers are all featured external Memory Controllers on the EBI. These external Memory Controllers are capable of handling several types of external memory and peripheral devices, such as SRAM, PROM, EPROM, EEPROM, Flash, and SDRAM.

The EBI also supports the CompactFlash and the NAND Flash protocols via integrated circuitry that greatly reduces the requirements for external components. Furthermore, the EBI handles data transfers with up to eight external devices, each assigned to eight address spaces defined by the embedded Memory Controller. Data transfers are performed through a 16-bit or 32-bit data bus, an address bus of up to 26 bits, up to eight chip select lines (NCS[7:0]) and several control pins that are generally multiplexed between the different external Memory Controllers.

15.2 Block Diagram

Figure 15-1 shows the organization of the External Bus Interface.

Figure 15-1. Organization of the External Bus Interface



15.3 I/O Lines Description

Table 15-1. I/O Lines Description

Name	Function	Type	Active Level
EBI			
D0 - D31	Data Bus	I/O	
A0 - A25	Address Bus	Output	
NWAIT	External Wait Signal	Input	Low
SMC			
NCS0 - NCS7	Chip Select Lines	Output	Low
NWR0 - NWR3	Write Signals	Output	Low
NRD	Read Signal	Output	Low
NWE	Write Enable	Output	Low
NBS0 - NBS3	Byte Mask Signals	Output	Low
EBI for CompactFlash Support			
CFCE1 - CFCE2	CompactFlash Chip Enable	Output	Low
CFOE	CompactFlash Output Enable	Output	Low
CFWE	CompactFlash Write Enable	Output	Low
CFIOR	CompactFlash I/O Read Signal	Output	Low
CFIOW	CompactFlash I/O Write Signal	Output	Low
CFRNW	CompactFlash Read Not Write Signal	Output	
CFCS0 - CFCS1	CompactFlash Chip Select Lines	Output	Low
EBI for NAND Flash Support			
NANDCS	NAND Flash Chip Select Line	Output	Low
NANDOE	NAND Flash Output Enable	Output	Low
NANDWE	NAND Flash Write Enable	Output	Low
SDRAM Controller			
SDCK	SDRAM Clock	Output	
SDCKE	SDRAM Clock Enable	Output	High
SDCS	SDRAM Controller Chip Select Line	Output	Low
BA0 - BA1	Bank Select	Output	
SDWE	SDRAM Write Enable	Output	Low
RAS - CAS	Row and Column Signal	Output	Low
NWR0 - NWR3	Write Signals	Output	Low
NBS0 - NBS3	Byte Mask Signals	Output	Low
SD_A10	SDRAM Address 10 Line	Output	

The connection of some signals through the MUX logic is not direct and depends on the Memory Controller in use at the moment.

Table 15-2 on page 133 details the connections between the two Memory Controllers and the EBI pins.

Table 15-2. EBI Pins and Memory Controllers I/O Lines Connections

EBI Pins	SDRAMC I/O Lines	SMC I/O Lines
NWR1/NBS1/CFIOR	NBS1	NWR1/NUB
A0/NBS0	Not Supported	SMC_A0/NLB
A1/NBS2/NWR2	Not Supported	SMC_A1
A[11:2]	SDRAMC_A[9:0]	SMC_A[11:2]
SD_A10	SDRAMC_A10	Not Supported
A12	Not Supported	SMC_A12
A[14:13]	SDRAMC_A[12:11]	SMC_A[14:13]
A[25:15]	Not Supported	SMC_A[25:15]
D[31:16]	D[31:16]	D[31:16]
D[15:0]	D[15:0]	D[15:0]

15.4 Application Example

15.4.1 Hardware Interface

Table 15-3 and Table 15-4 detail the connections to be applied between the EBI pins and the external devices for each Memory Controller.

Table 15-3. EBI Pins and External Static Devices Connections

	Pins of the Interfaced Device					
Pins	8-bit Static Device	2 x 8-bit Static Devices	16-bit Static Device	4 x 8-bit Static Devices	2 x 16-bit Static Devices	32-bit Static Device
Controller	**SMC**					
D0 - D7	D0 - D7	D0 - D7	D0 - D7	D0 - D7	D0 - D7	D0 - D7
D8 - D15	–	D8 - D15	D8 - D15	D8 - D15	D8 - 15	D8 - 15
D16 - D23	–	–	–	D16 - D23	D16 - D23	D16 - D23
D24 - D31	–	–	–	D24 - D31	D24 - D31	D24 - D31
A0/NBS0	A0	–	NLB	–	NLB(3)	BE0(5)
A1/NWR2/NBS2	A1	A0	A0	WE(2)	NLB(4)	BE2(5)
A2 - A25	A[2:25]	A[1:24]	A[1:24]	A[0:23]	A[0:23]	A[0:23]
NCS0	CS	CS	CS	CS	CS	CS
NCS1/SDCS	CS	CS	CS	CS	CS	CS
NCS2	CS	CS	CS	CS	CS	CS
NCS3/NANDCS	CS	CS	CS	CS	CS	CS
NCS4/CFCS0	CS	CS	CS	CS	CS	CS

Table 15-3. EBI Pins and External Static Devices Connections (Continued)

Pins	Pins of the Interfaced Device					
	8-bit Static Device	2 x 8-bit Static Devices	16-bit Static Device	4 x 8-bit Static Devices	2 x 16-bit Static Devices	32-bit Static Device
Controller	**SMC**					
NCS5/CFCS1	CS	CS	CS	CS	CS	CS
NCS6/NANDOE	CS	CS	CS	CS	CS	CS
NCS7/NANDWE	CS	CS	CS	CS	CS	CS
NRD/CFOE	OE	OE	OE	OE	OE	OE
NWR0/NWE	WE	WE(1)	WE	WE(2)	WE	WE
NWR1/NBS1	–	WE(1)	NUB	WE(2)	NUB(3)	BE1(5)
NWR3/NBS3	–	–	–	WE(2)	NUB(4)	BE3(5)

Notes:
1. NWR1 enables upper byte writes. NWR0 enables lower byte writes.
2. NWRx enables corresponding byte x writes. (x = 0, 1, 2 or 3)
3. NBS0 and NBS1 enable respectively lower and upper bytes of the lower 16-bit word.
4. NBS2 and NBS3 enable respectively lower and upper bytes of the upper 16-bit word.
5. BEx: Byte x Enable (x = 0,1,2 or 3)

Table 15-4. EBI Pins and External Devices Connections

Pins	Pins of the Interfaced Device			
	SDRAM	Compact Flash	Compact Flash True IDE Mode	NAND Flash
Controller	**SDRAMC**	**SMC**		
D0 - D7	D0 - D7	D0 - D7	D0 - D7	I/O0-I/O7
D8 - D15	D8 - D15	D8 - 15	D8 - 15	I/O8-I/O15
D16 - D31	D16 - D31	–	–	–
A0/NBS0	DQM0	A0	A0	–
A1/NWR2/NBS2	DQM2	A1	A1	–
A2 - A10	A[0:8]	A[2:10]	A[2:10]	–
A11	A9	–	–	–
SD_A10	A10	–	–	–
A12	–	–	–	–
A13 - A14	A[11:12]	–	–	–
A15	–	–	–	–
A16/BA0	BA0	–	–	–
A17/BA1	BA1	–	–	–
A18 - A20	–	–	–	–
A21	–	–	–	CLE
A22	–	REG	REG	ALE
A23 - A24	–	–	–	–

Table 15-4. EBI Pins and External Devices Connections (Continued)

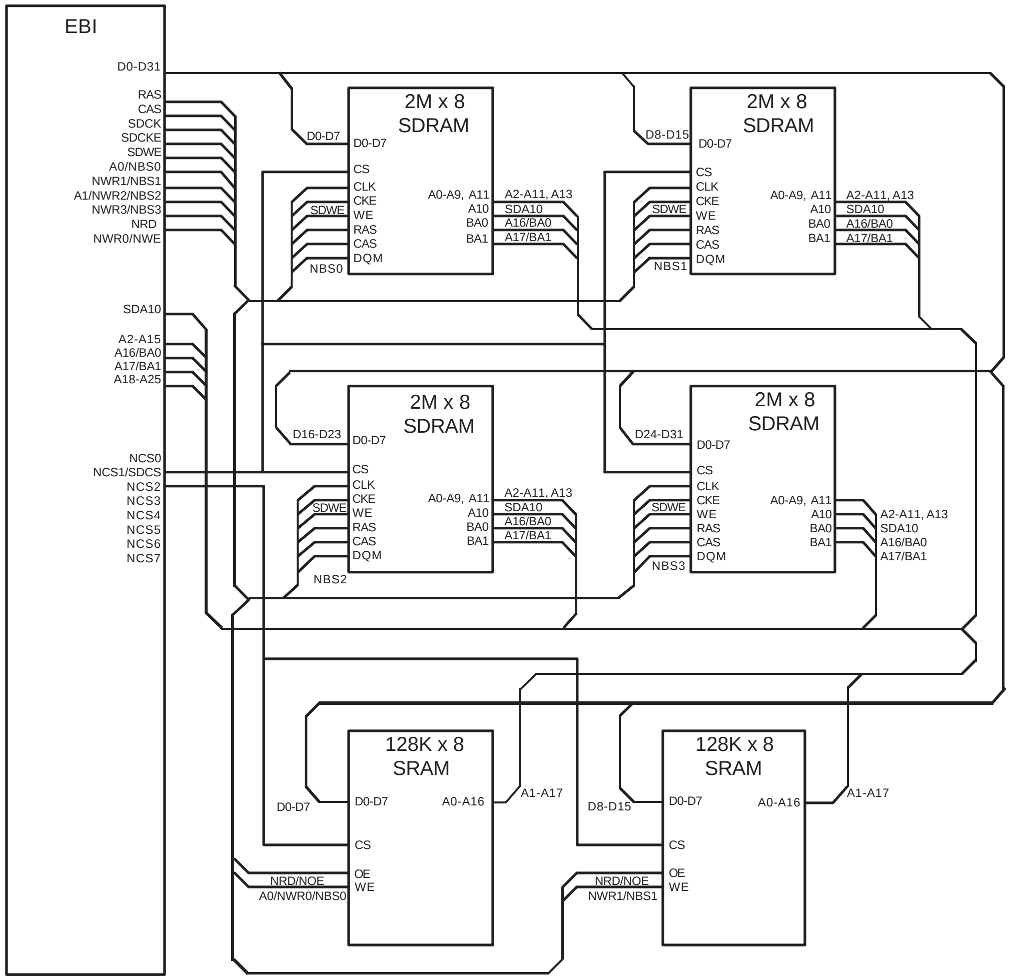
Pins	Pins of the Interfaced Device			
	SDRAM	Compact Flash	Compact Flash True IDE Mode	NAND Flash
Controller	**SDRAMC**	**SMC**		
A25	–	CFRNW(1)	CFRNW(1)	–
NCS0	–	–	–	–
NCS1/SDCS	CS	–	–	–
NCS2	–	–	–	–
NCS3/NANDCS	–	–	–	CE(3)
NCS4/CFCS0	–	CFCS0(1)	CFCS0(1)	–
NCS5/CFCS1	–	CFCS1(1)	CFCS1(1)	–
NCS6/NANDOE	–	–	–	RE
NCS7/NANDWE	–	–	–	WE
NRD/CFOE	–	OE	–	–
NWR0/NWE/CFWE	–	WE	WE	–
NWR1/NBS1/CFIOR	DQM1	IOR	IOR	–
NWR3/NBS3/CFIOW	DQM3	IOW	IOW	–
CFCE1	–	CE1	CS0	–
CFCE2	–	CE2	CS1	–
SDCK	CLK	–	–	–
SDCKE	CKE	–	–	–
RAS	RAS	–	–	–
CAS	CAS	–	–	–
SDWE	WE	–	–	–
NWAIT	–	WAIT	WAIT	–
Pxx(2)	–	CD1 or CD2	CD1 or CD2	–
Pxx(2)	–	–	–	CE(3)
Pxx(2)	–	–	–	RDY

Notes: 1. Not directly connected to the CompactFlash slot. Permits the control of the bidirectional buffer between the EBI data bus and the CompactFlash slot.
2. Any PIO line.
3. CE connection depends on the NAND Flash. For standard NAND Flash devices, it must be connected to any free PIO line. For “CE don’t care” NAND Flash devices, it can be connected to either NCS3/NANDCS or to any free PIO line.

15.4.2 Connection Examples

Figure 15-2 shows an example of connections between the EBI and external devices.

Figure 15-2. EBI Connections to Memory Devices



15.5 Product Dependencies

15.5.1 I/O Lines

The pins used for interfacing the External Bus Interface may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the External Bus Interface pins to their peripheral function. If I/O lines of the External Bus Interface are not used by the application, they can be used for other purposes by the PIO Controller.

15.6 Functional Description

The EBI transfers data between the internal AHB Bus (handled by the Bus Matrix) and the external memories or peripheral devices. It controls the waveforms and the parameters of the external address, data and control busses and is composed of the following elements:

- Static Memory Controller (SMC)
- SDRAM Controller (SDRAMC)
- A chip select assignment feature that assigns an AHB address space to the external devices
- A multiplex controller circuit that shares the pins between the different Memory Controllers
- Programmable CompactFlash support logic
- Programmable NAND Flash support logic

15.6.1 Bus Multiplexing

The EBI offers a complete set of control signals that share the 32-bit data lines, the address lines of up to 26 bits and the control signals through a multiplex logic operating in function of the memory area requests.

Multiplexing is specifically organized in order to guarantee the maintenance of the address and output control lines at a stable state while no external access is being performed. Multiplexing is also designed to respect the data float times defined in the Memory Controllers. Furthermore, refresh cycles of the SDRAM are executed independently by the SDRAM Controller without delaying the other external Memory Controller accesses.

15.6.2 Pull-up Control

The EBI_CSA register in the Bus Matrix User Interface permits enabling of on-chip pull-up resistors on the data bus lines not multiplexed with the PIO Controller lines. The pull-up resistors are enabled after reset. Setting the DBPUC bit disables the pull-up resistors on the D0 to D15 lines. Enabling the pull-up resistor on the D16-D31 lines can be performed by programming the appropriate PIO controller.

15.6.3 Static Memory Controller

For information on the Static Memory Controller, refer to the Static Memory Controller section.

15.6.4 SDRAM Controller

For information on the SDRAM Controller, refer to the SDRAM section.

15.6.5 CompactFlash Support

The External Bus Interface integrates circuitry that interfaces to CompactFlash devices.

The CompactFlash logic is driven by the Static Memory Controller (SMC) on the NCS4 and/or NCS5 address space. Programming the CS4A and/or CS5A bit of the EBI_CSA Register to the appropriate value enables this logic. For details on this register, refer to the Bus Matrix User Interface section. Access to an external CompactFlash device is then made by accessing the address space reserved to NCS4 and/or NCS5 (i.e., between 0x5000 0000 and 0x5FFF FFFF for NCS4 and between 0x6000 0000 and 0x6FFF FFFF for NCS5).

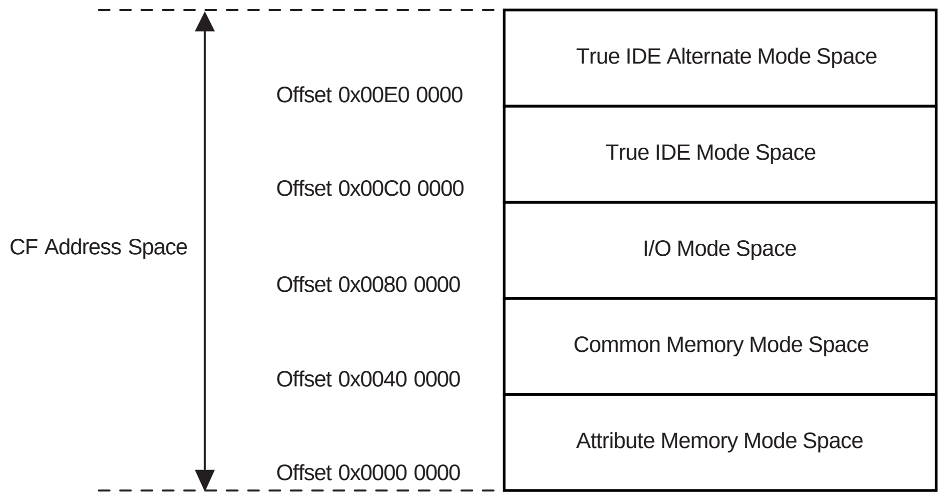
All CompactFlash modes (Attribute Memory, Common Memory, I/O and True IDE) are supported but the signals _IOIS16 (I/O and True IDE modes) and _ATA SEL (True IDE mode) are not handled.

15.6.5.1 *I/O Mode, Common Memory Mode, Attribute Memory Mode and True IDE Mode*

Within the NCS4 and/or NCS5 address space, the current transfer address is used to distinguish I/O mode, common memory mode, attribute memory mode and True IDE mode.

The different modes are accessed through a specific memory mapping as illustrated on Figure 15-3. A[23:21] bits of the transfer address are used to select the desired mode as described in Table 15-5 on page 138.

Figure 15-3. CompactFlash Memory Mapping



Note: The A22 pin of the EBI is used to drive the REG signal of the CompactFlash Device (except in True IDE mode).

Table 15-5. CompactFlash Mode Selection

A[23:21]	Mode Base Address
000	Attribute Memory
010	Common Memory
100	I/O Mode
110	True IDE Mode
111	Alternate True IDE Mode

15.6.5.2 *CFCE1 and CFCE2 signals*

To cover all types of access, the SMC must be alternatively set to drive 8-bit data bus or 16-bit data bus. The odd byte access on the D[7:0] bus is only possible when the SMC is configured to drive 8-bit memory devices on the corresponding NCS pin (NCS4 and or NCS5). The Chip Select Register (DBW field in the corresponding Chip Select Mode Register) of the NCS4 and/or NCS5 address space must be set as shown in Table 15-6 to enable the required access type.

NBS1 and NBS0 are the byte selection signals from SMC and are available when the SMC is set in Byte Select mode on the corresponding Chip Select.

The CFCE1 and CFCE2 waveforms are identical to the corresponding NCSx waveform. For details on these waveforms and timings, refer to the Static Memory Controller section.

Table 15-6. CFCE1 and CFCE2 Truth Table

Mode	CFCE2	CFCE1	DBW	Comment	SMC Access Mode
Attribute Memory	NBS1	NBS0	16 bits	Access to Even Byte on D[7:0]	Byte Select
Common Memory	NBS1	NBS0	16bits	Access to Even Byte on D[7:0] Access to Odd Byte on D[15:8]	Byte Select
	1	0	8 bits	Access to Odd Byte on D[7:0]	Don't Care
I/O Mode	NBS1	NBS0	16 bits	Access to Even Byte on D[7:0] Access to Odd Byte on D[15:8]	Byte Select
	1	0	8 bits	Access to Odd Byte on D[7:0]	Don't Care
True IDE Mode					
Task File	1	0	8 bits	Access to Even Byte on D[7:0] Access to Odd Byte on D[7:0]	Don't Care
Data Register	1	0	16 bits	Access to Even Byte on D[7:0] Access to Odd Byte on D[15:8]	Byte Select
Alternate True IDE Mode					
Control Register Alternate Status Read	0	1	Don't Care	Access to Even Byte on D[7:0]	Don't Care
Drive Address	0	1	8 bits	Access to Odd Byte on D[7:0]	Don't Care
True IDE Standby Mode or Address Space is not assigned to CF	1	1	Don't Care	Don't Care	Don't Care

15.6.5.3 Read/Write Signals

In I/O mode and True IDE mode, the CompactFlash logic drives the read and write command signals of the SMC on CFIOR and CFIOW signals, while the CFOE and CFWE signals are deactivated. Likewise, in common memory mode and attribute memory mode, the SMC signals are driven on the CFOE and CFWE signals, while the CFIOR and CFIOW are deactivated. Figure 15-4 on page 140 demonstrates a schematic representation of this logic.

Attribute memory mode, common memory mode and I/O mode are supported by setting the address setup and hold time on the NCS4 (and/or NCS5) chip select to the appropriate values.

Figure 15-4. CompactFlash Read/Write Control Signals

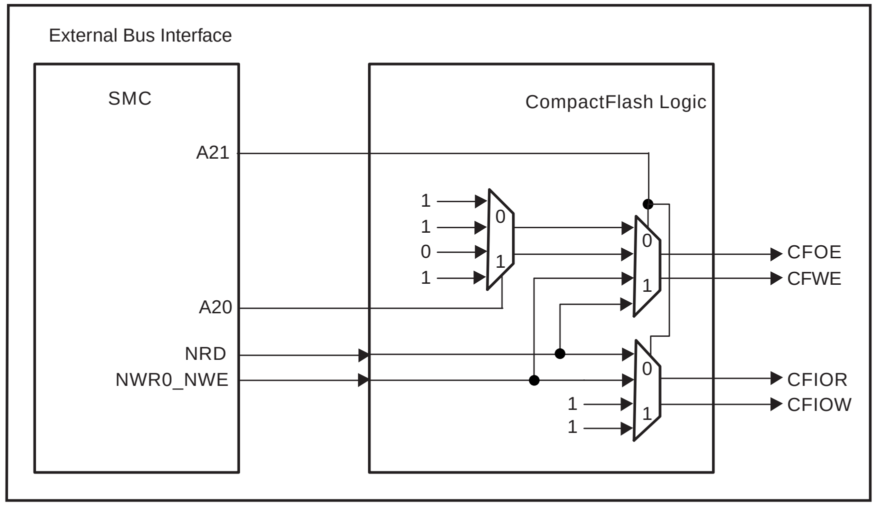


Table 15-7. CompactFlash Mode Selection

Mode Base Address	CFOE	CFWE	CFIOR	CFIOW
Attribute Memory Common Memory	NRD	NWR0_NWE	1	1
I/O Mode	1	1	NRD	NWR0_NWE
True IDE Mode	0	1	NRD	NWR0_NWE

15.6.5.4 Multiplexing of CompactFlash Signals on EBI Pins

Table 15-8 on page 140 and Table 15-9 on page 141 illustrate the multiplexing of the CompactFlash logic signals with other EBI signals on the EBI pins. The EBI pins in Table 15-8 are strictly dedicated to the CompactFlash interface as soon as the CS4A and/or CS5A field of the EBI_CSA Register is set. These pins must not be used to drive any other memory devices.

The EBI pins in Table 15-9 on page 141 remain shared between all memory areas when the corresponding CompactFlash interface is enabled (CS4A = 1 and/or CS5A = 1).

Table 15-8. Dedicated CompactFlash Interface Multiplexing

Pins	CompactFlash Signals		EBI Signals	
	CS4A = 1	**CS5A = 1**	**CS4A = 0**	**CS5A = 0**
NCS4/CFCS0	CFCS0		NCS4	
NCS5/CFCS1		CFCS1		NCS5

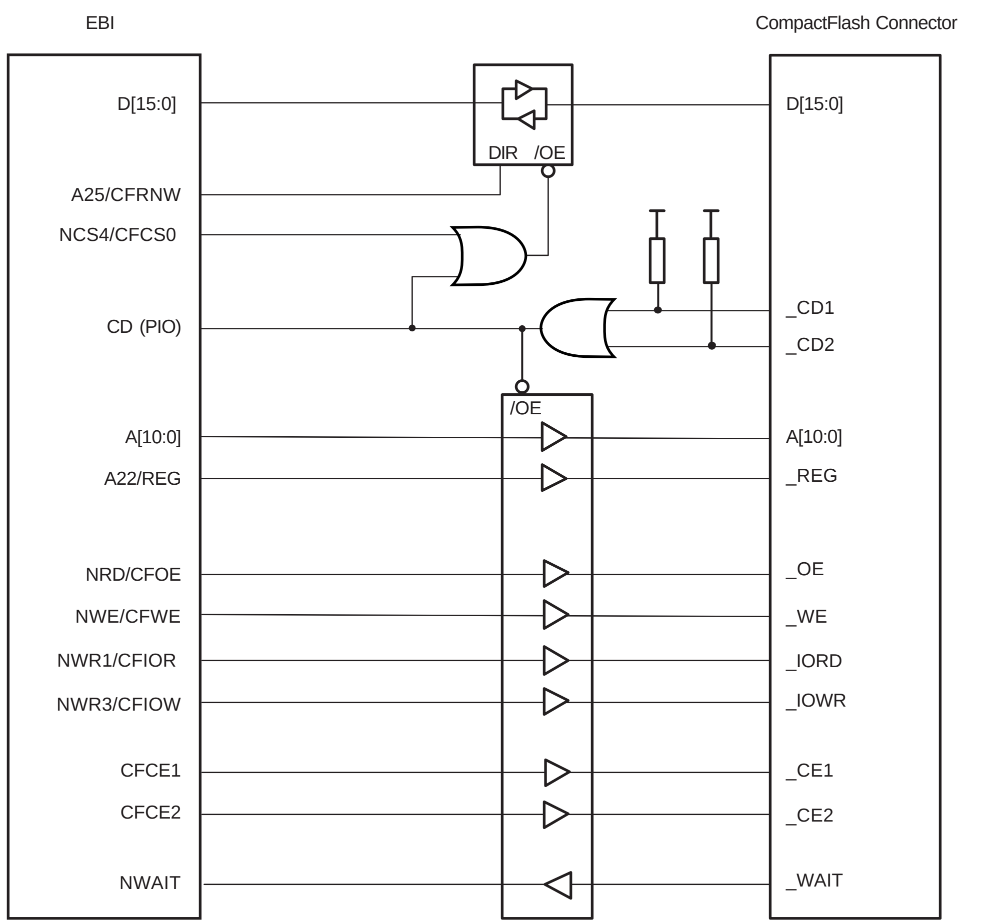
Table 15-9. Shared CompactFlash Interface Multiplexing

Pins	Access to CompactFlash Device	Access to Other EBI Devices
	CompactFlash Signals	**EBI Signals**
NRD/CFOE	CFOE	NRD
NWR0/NWE/CFWE	CFWE	NWR0/NWE
NWR1/NBS1/CFIOR	CFIOR	NWR1/NBS1
NWR3/NBS3/CFIOW	CFIOW	NWR3/NBS3
A25/CFRNW	CFRNW	A25

15.6.5.5 Application Example

Figure 15-5 on page 141 illustrates an example of a CompactFlash application.CFCS0 and CFRNW signals are not directly connected to the CompactFlash slot 0, but do control the direction and the output enable of the buffers between the EBI and the CompactFlash Device. The timing of the CFCS0 signal is identical to the NCS4 signal. Moreover, the CFRNW signal remains valid throughout the transfer, as does the address bus. The CompactFlash _WAIT signal is connected to the NWAIT input of the Static Memory Controller. For details on these waveforms and timings, refer to the Static Memory Controller section.

Figure 15-5. CompactFlash Application Example



15.6.6 NAND Flash Support

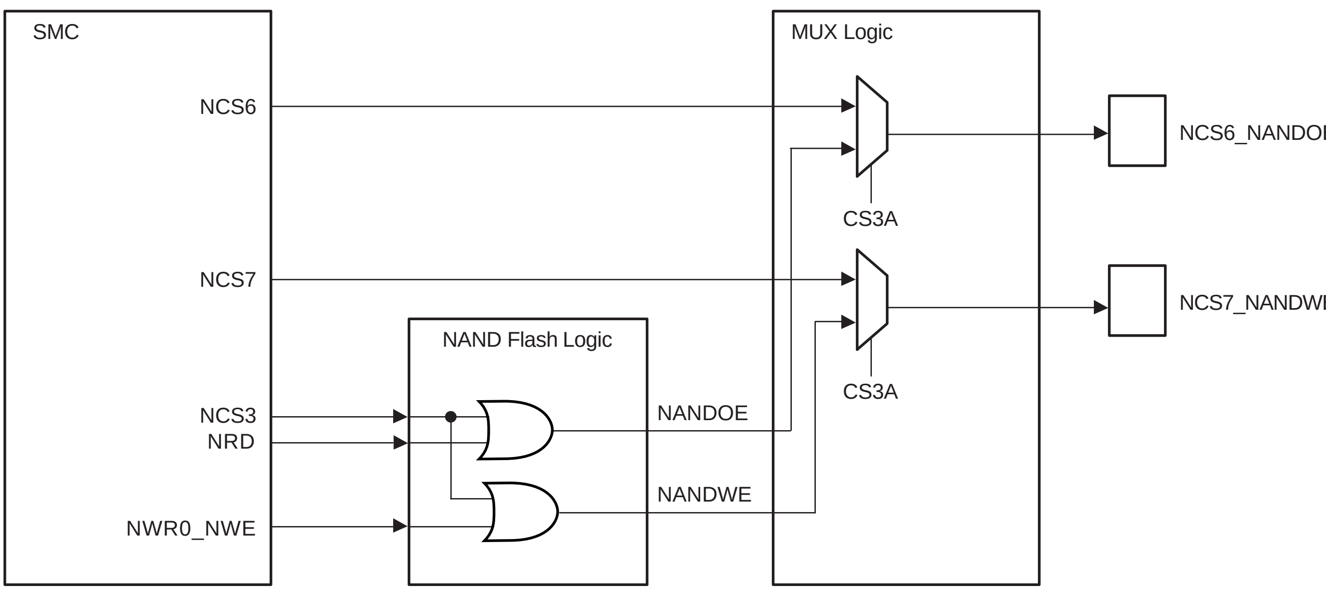
The EBI integrates circuitry that interfaces to NAND Flash devices.

The NAND Flash logic is driven by the Static Memory Controller on the NCS3 address space. Programming the CS3A field in the EBI_CSA Register in the Bus Matrix User Interface to the appropriate value enables the NAND Flash logic. For details on this register, refer to the Bus Matrix User Interface section. Access to an external NAND Flash device is then made by accessing the address space reserved to NCS3 (i.e., between 0x40000000 and 0x4FFF FFFF).

The NAND Flash Logic drives the read and write command signals of the SMC on the NANDOE and NANDWE signals when the NCS3 signal is active. NANDOE and NANDWE are invalidated as soon as the transfer address fails to lie in the NCS3 address space. For details on these waveforms, refer to the Static Memory Controller section.

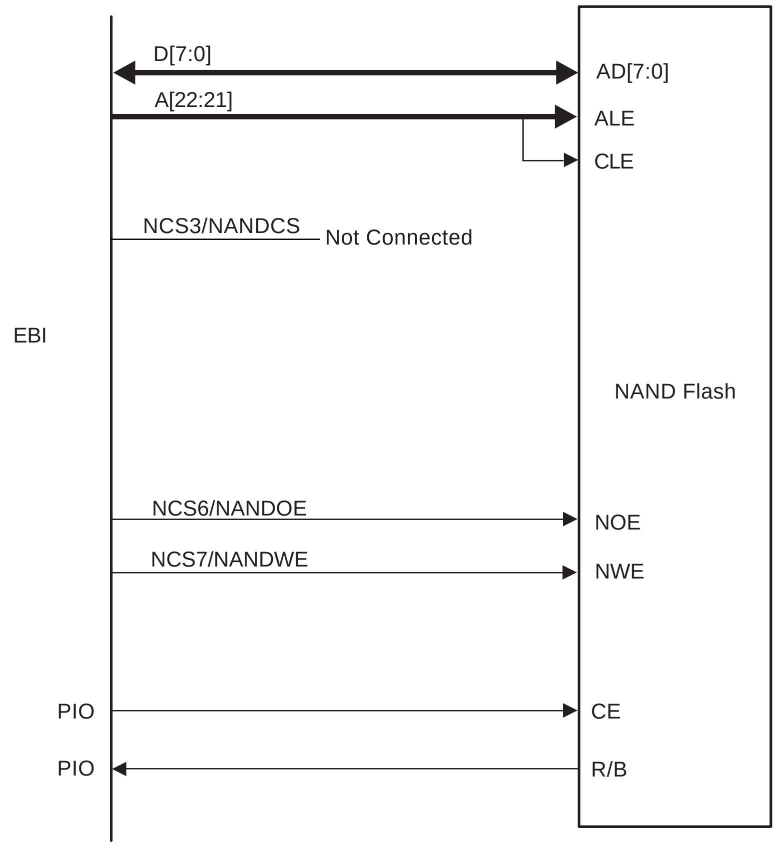
The NANDOE and NANDWE signals are multiplexed with NCS6 and NCS7 signals of the Static Memory Controller. This multiplexing is controlled in the MUX logic part of the EBI by the CS3A bit in the in the EBI_CSA Register For details on this register, refer to the Bus Matrix User Interface Section. NCS6 and NCS7 become unavailable. Performing an access within the address space reserved to NCS6 and NCS7 (i.e., between 0x70000000 and 0x8FFF FFFF) may lead to an unpredictable outcome.

Figure 15-6. NAND Flash Signal Multiplexing on EBI Pins



The address latch enable and command latch enable signals on the NAND Flash device are driven by address bits A22 and A21 of the EBI address bus. The user should note that any bit on the EBI address bus can also be used for this purpose. The command, address or data words on the data bus of the NAND Flash device are distinguished by using their address within the NCS3 address space. The chip enable (CE) signal of the device and the ready/busy (R/B) signals are connected to PIO lines. The CE signal then remains asserted even when NCS3 is not selected, preventing the device from returning to standby mode.

Figure 15-7. NAND Flash Application Example



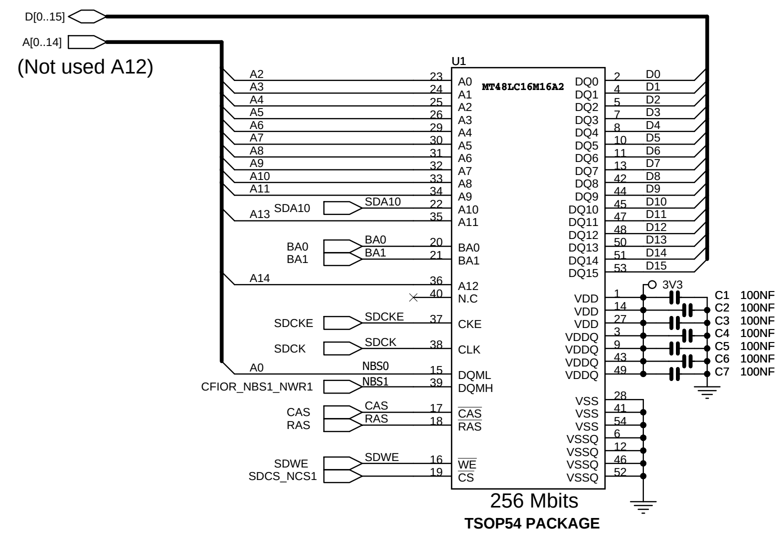
Note: The External Bus Interface is also able to support 16-bits devices.

15.7 Implementation Examples

All the hardware configurations are given for illustration only. The user should refer to the memory manufacturer web site to check the device availability.

15.7.1 16-bit SDRAM

15.7.1.1 Hardware Configuration



15.7.1.2 Software Configuration

The following configuration has to be performed:

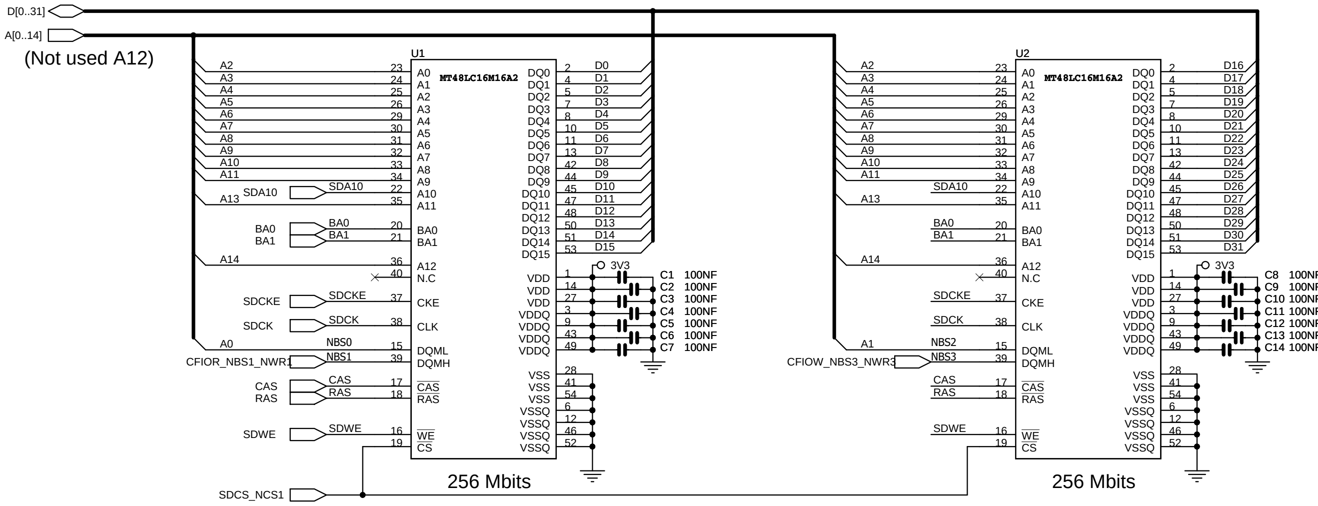
- Assign the EBI NCS1 to the SDRAM controller by setting the bit EBI_CS1A in the EBI Chip Select Assignment Register located in the bus matrix memory space.
- Initialize the SDRAM Controller depending on the SDRAM device and system bus frequency.

The Data Bus Width is to be programmed to 16 bits.

The SDRAM initialization sequence is described in the “SDRAM device initialisation” part of the SDRAM controller.

15.7.2 32-bit SDRAM

15.7.2.1 Hardware Configuration



TSOP54 PACKAGE

15.7.2.2 Software Configuration

The following configuration has to be performed:

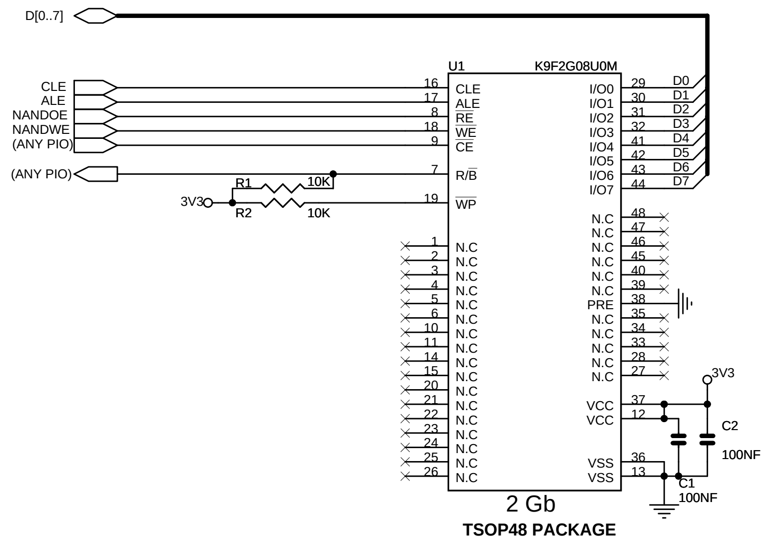
- Assign the EBI NCS1 to the SDRAM controller by setting the bit EBI_CS1A in the EBI Chip Select Assignment Register located in the bus matrix memory space.
- Initialize the SDRAM Controller depending on the SDRAM device and system bus frequency.

The Data Bus Width is to be programmed to 32 bits. The data lines D[16..31] are multiplexed with PIO lines and thus the dedicated PIOs must be programmed in peripheral mode in the PIO controller.

The SDRAM initialization sequence is described in the “SDRAM device initialisation” part of the SDRAM controller.

15.7.3 8-bit NANDFlash

15.7.3.1 Hardware Configuration



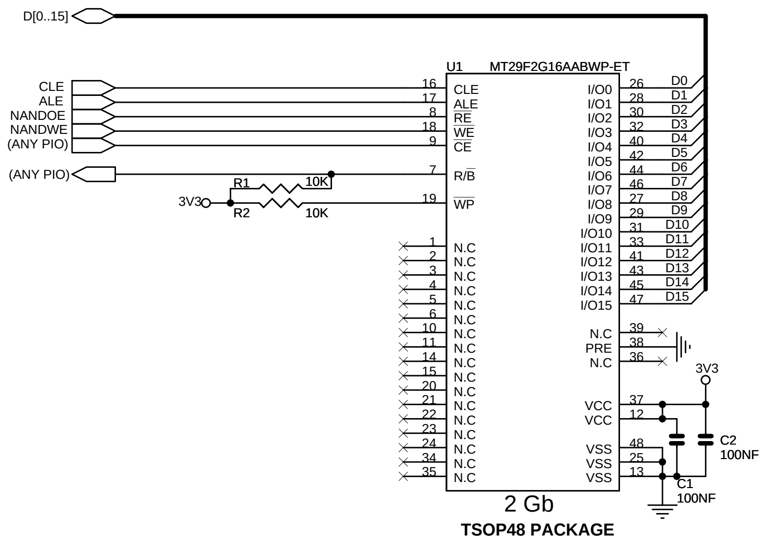
15.7.3.2 Software Configuration

The following configuration has to be performed:

- Assign the EBI CS3 to the NandFlash by setting the bit EBI_CS3A in the EBI Chip Select Assignment Register located in the bus matrix memory space
- Reserve A21 / A22 for ALE / CLE functions. Address and Command Latches are controlled respectively by setting to 1 the address bit A21 and A22 during accesses.
- NANDOE and NANDWE signals are multiplexed with PIO lines and thus the dedicated PIOs must be programmed in peripheral mode in the PIO controller.
- Configure a PIO line as an input to manage the Ready/Busy signal.
- Configure Static Memory Controller CS3 Setup, Pulse, Cycle and Mode accordingly to NANDFlash timings, the data bus width and the system bus frequency.

15.7.4 16-bit NANDFlash

15.7.4.1 Hardware Configuration

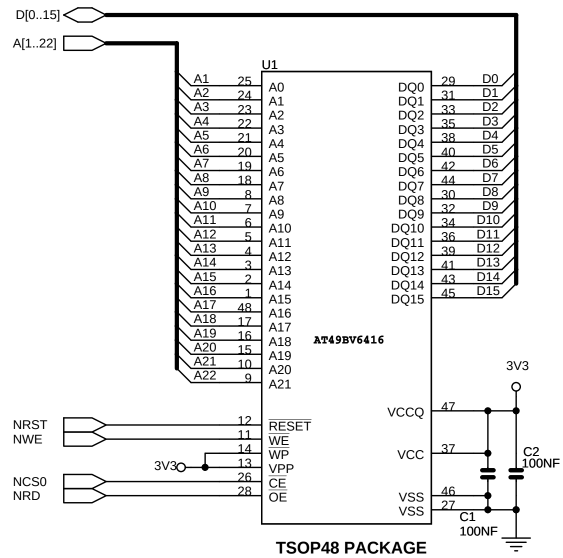


15.7.4.2 Software Configuration

The software configuration is the same as for an 8-bit NandFlash except the data bus width programmed in the mode register of the Static Memory Controller.

15.7.5 NOR Flash on NCS0

15.7.5.1 Hardware Configuration



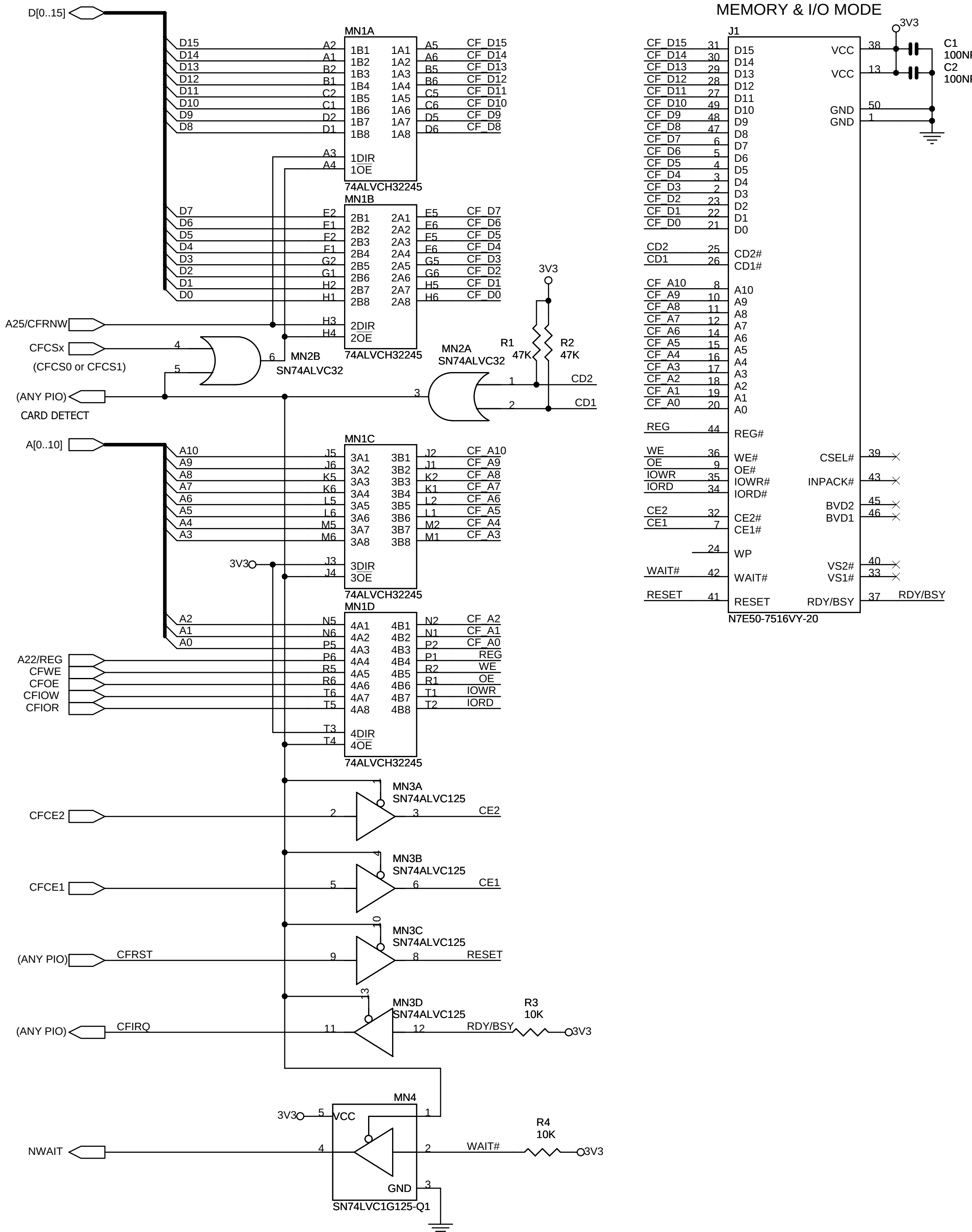
15.7.5.2 Software Configuration

The default configuration for the Static Memory Controller, byte select mode, 16-bit data bus, Read/Write controlled by Chip Select, allows boot on 16-bit non-volatile memory at slow clock.

For another configuration, configure the Static Memory Controller NCS0 Setup, Pulse, Cycle and Mode depending on Flash timings and system bus frequency.

15.7.6 Compact Flash

15.7.6.1 Hardware Configuration



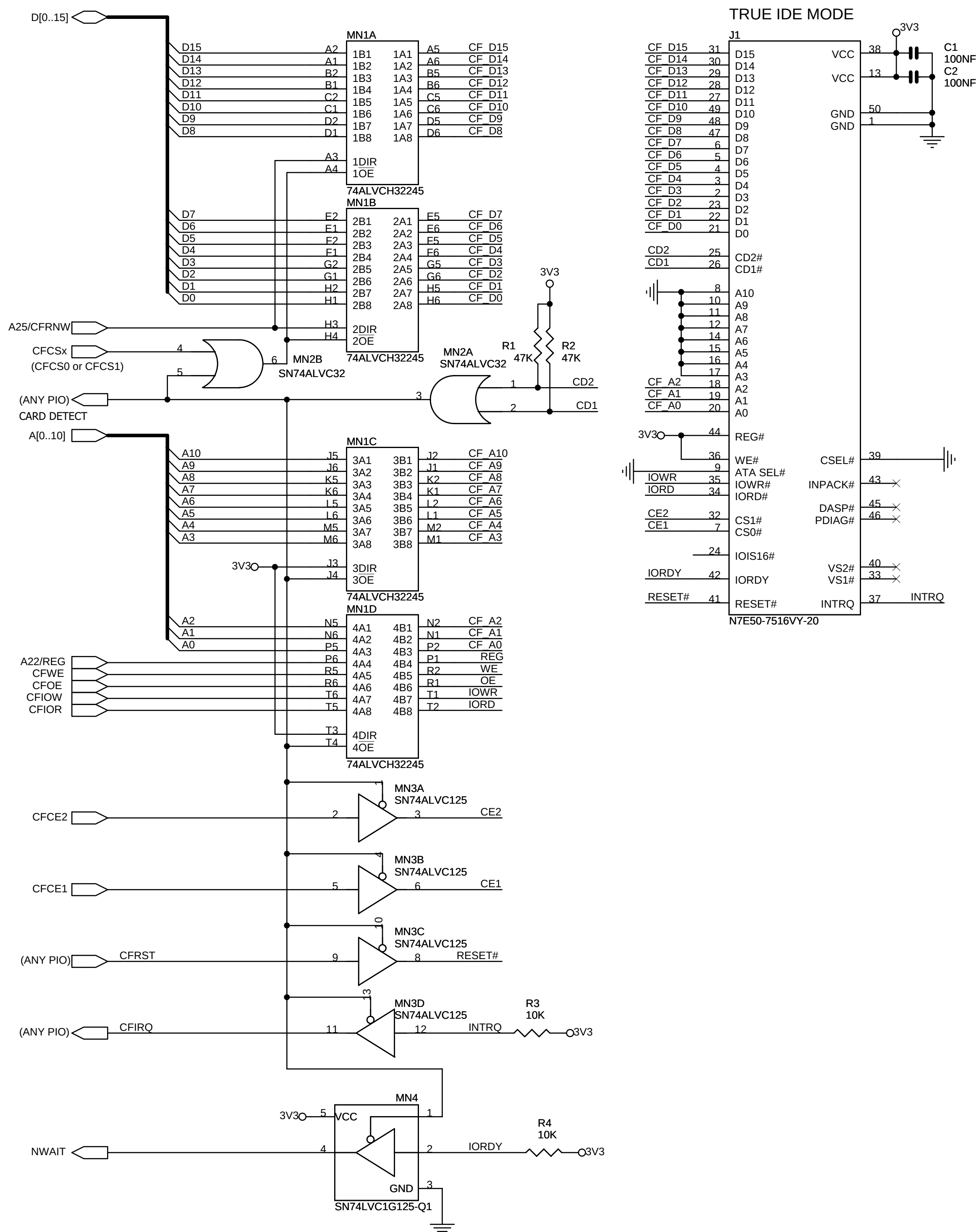
15.7.6.2 Software Configuration

The following configuration has to be performed:

- Assign the EBI NCS4 and/or EBI NCS5 to the CompactFlash Slot 0 or/and Slot 1 by setting the bit EBI_CS4A or/and EBI_CS5A in the EBI Chip Select Assignment Register located in the bus matrix memory space.
- The address line A23 is to select I/O (A23=1) or Memory mode (A23=0) and the address line A22 for REG function.
- A23, CFRNW, CFS0, CFCS1, CFCE1 and CFCE2 signals are multiplexed with PIO lines and thus the dedicated PIOs must be programmed in peripheral mode in the PIO controller.
- Configure a PIO line as an output for CFRST and two others as an input for CFIRQ and CARD DETECT functions respectively.
- Configure SMC NCS4 and/or SMC NCS5 (for Slot 0 or 1) Setup, Pulse, Cycle and Mode accordingly to Compact Flash timings and system bus frequency.

15.7.7 Compact Flash True IDE

15.7.7.1 Hardware Configuration



15.7.7.2 Software Configuration

The following configuration has to be performed:

- Assign the EBI NCS4 and/or EBI NCS5 to the CompactFlash Slot 0 or/and Slot 1 by setting the bit EBI_CS4A or/and EBI_CS5A in the EBI Chip Select Assignment Register located in the bus matrix memory space.
- The address line A21 is to select Alternate True IDE (A21=1) or True IDE (A21=0) modes.
- CFRNW, CFS0, CFCS1, CFCE1 and CFCE2 signals are multiplexed with PIO lines and thus the dedicated PIOs must be programmed in peripheral mode in the PIO controller.
- Configure a PIO line as an output for CFRST and two others as an input for CFIRQ and CARD DETECT functions respectively.
- Configure SMC NCS4 and/or SMC NCS5 (for Slot 0 or 1) Setup, Pulse, Cycle and Mode accordingly to Compact Flash timings and system bus frequency.

16. Static Memory Controller (SMC)

16.1 Description

The Static Memory Controller (SMC) generates the signals that control the access to the external memory devices or peripheral devices. It has 8 Chip Selects and a 26-bit address bus. The 32-bit data bus can be configured to interface with 8-, 16-, or 32-bit external devices. Separate read and write control signals allow for direct memory and peripheral interfacing. Read and write signal waveforms are fully parametrizable.

The SMC can manage wait requests from external devices to extend the current access. The SMC is provided with an automatic slow clock mode. In slow clock mode, it switches from user-programmed waveforms to slow-rate specific waveforms on read and write signals. The SMC supports asynchronous burst read in page mode access for page size up to 32 bytes.

16.2 I/O Lines Description

Table 16-1. I/O Line Description

Name	Description	Type	Active Level
NCS[7:0]	Static Memory Controller Chip Select Lines	Output	Low
NRD	Read Signal	Output	Low
NWR0/NWE	Write 0/Write Enable Signal	Output	Low
A0/NBS0	Address Bit 0/Byte 0 Select Signal	Output	Low
NWR1/NBS1	Write 1/Byte 1 Select Signal	Output	Low
A1/NWR2/NBS2	Address Bit 1/Write 2/Byte 2 Select Signal	Output	Low
NWR3/NBS3	Write 3/Byte 3 Select Signal	Output	Low
A[25:2]	Address Bus	Output	
D[31:0]	Data Bus	I/O	
NWAIT	External Wait Signal	Input	Low

16.3 Multiplexed Signals

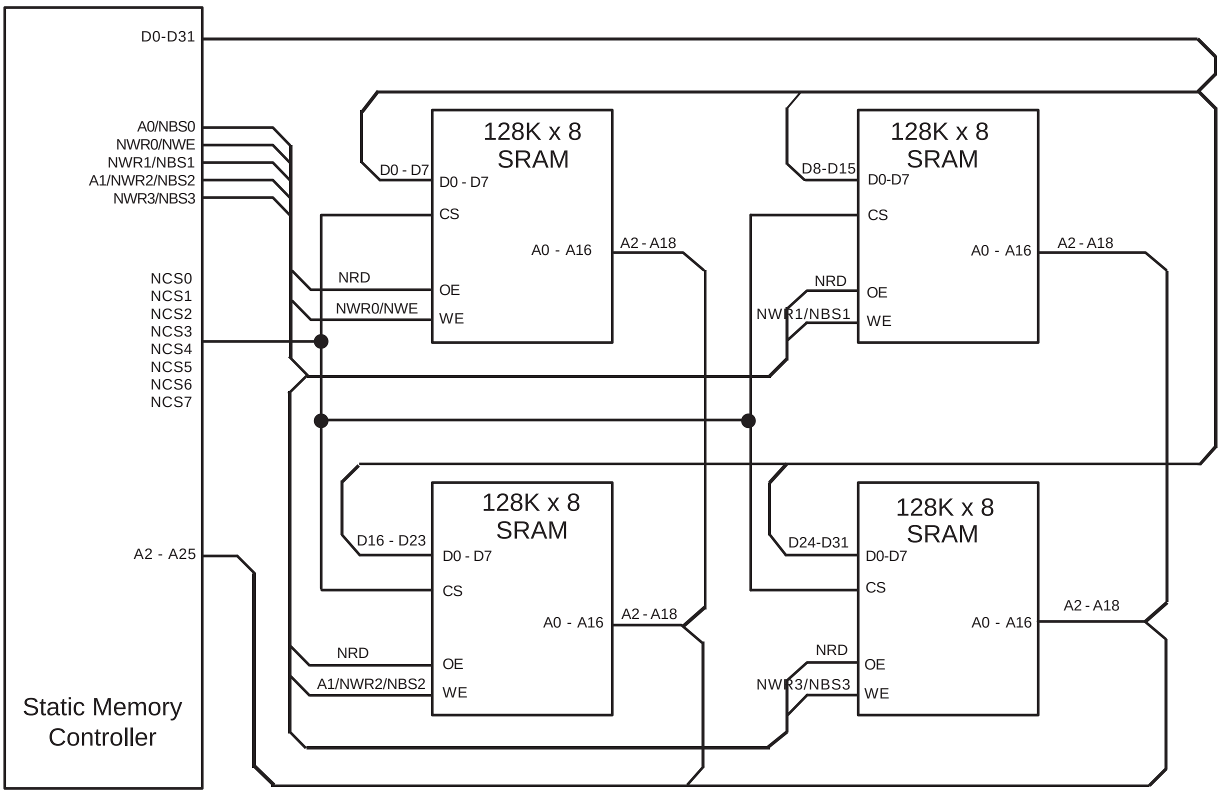
Table 16-2. Static Memory Controller (SMC) Multiplexed Signals

Multiplexed Signals			Related Function
NWR0	NWE		Byte-write or byte-select access, see "Byte Write or Byte Select Access" on page 155
A0	NBS0		8-bit or 16-/32-bit data bus, see "Data Bus Width" on page 155
NWR1	NBS1		Byte-write or byte-select access see "Byte Write or Byte Select Access" on page 155
A1	NWR2	NBS2	8-/16-bit or 32-bit data bus, see "Data Bus Width" on page 155. Byte-write or byte-select access, see "Byte Write or Byte Select Access" on page 155
NWR3	NBS3		Byte-write or byte-select access see "Byte Write or Byte Select Access" on page 155

16.4 Application Example

16.4.1 Hardware Interface

Figure 16-1. SMC Connections to Static Memory Devices



16.5 Product Dependencies

16.5.1 I/O Lines

The pins used for interfacing the Static Memory Controller may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the Static Memory Controller pins to their peripheral function. If I/O Lines of the SMC are not used by the application, they can be used for other purposes by the PIO Controller.

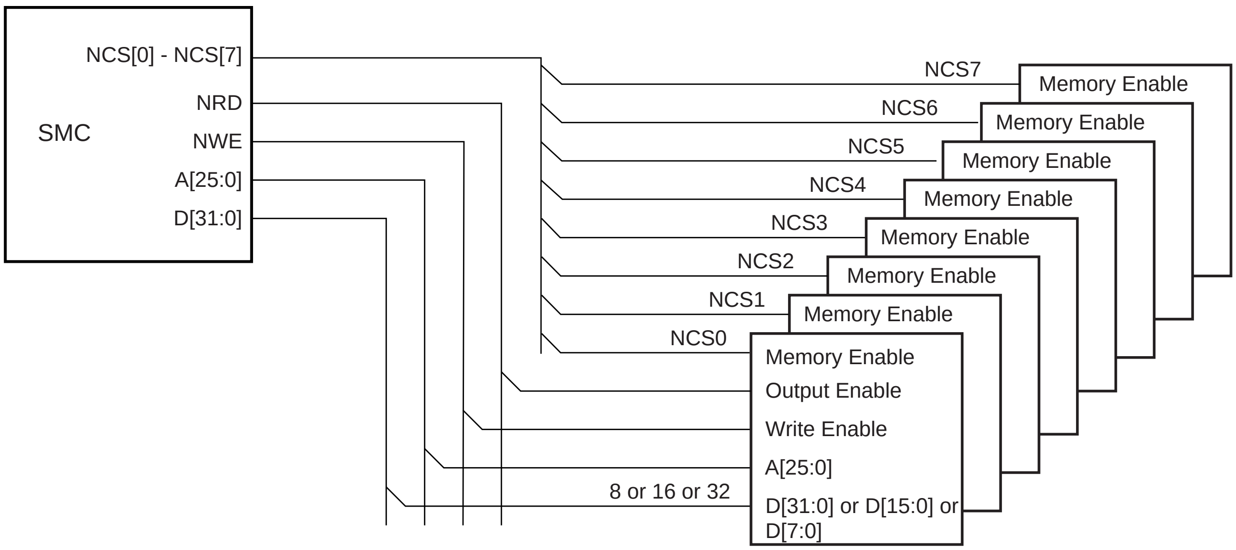
16.6 External Memory Mapping

The SMC provides up to 26 address lines, A[25:0]. This allows each chip select line to address up to 64 Mbytes of memory.

If the physical memory device connected on one chip select is smaller than 64 Mbytes, it wraps around and appears to be repeated within this space. The SMC correctly handles any valid access to the memory device within the page (see Figure 16-2).

A[25:0] is only significant for 8-bit memory, A[25:1] is used for 16-bit memory, A[25:2] is used for 32-bit memory.

Figure 16-2. Memory Connections for Eight External Devices



16.7 Connection to External Devices

16.7.1 Data Bus Width

A data bus width of 8, 16, or 32 bits can be selected for each chip select. This option is controlled by the field DBW in SMC_MODE (Mode Register) for the corresponding chip select.

Figure 16-3 shows how to connect a 512K x 8-bit memory on NCS2. Figure 16-4 shows how to connect a 512K x 16-bit memory on NCS2. Figure 16-5 shows two 16-bit memories connected as a single 32-bit memory

16.7.2 Byte Write or Byte Select Access

Each chip select with a 16-bit or 32-bit data bus can operate with one of two different types of write access: byte write or byte select access. This is controlled by the BAT field of the SMC_MODE register for the corresponding chip select.

Figure 16-3. Memory Connection for an 8-bit Data Bus

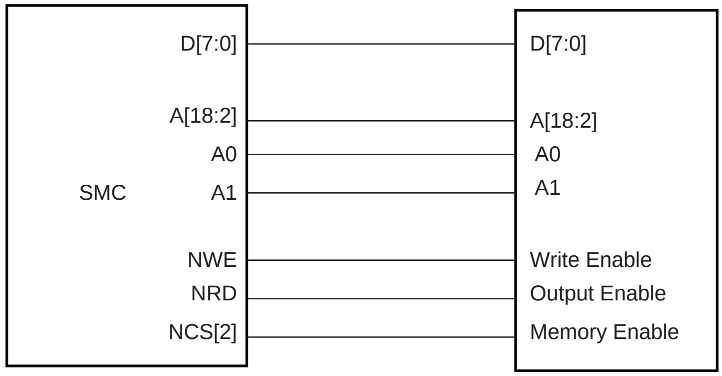


Figure 16-4. Memory Connection for a 16-bit Data Bus

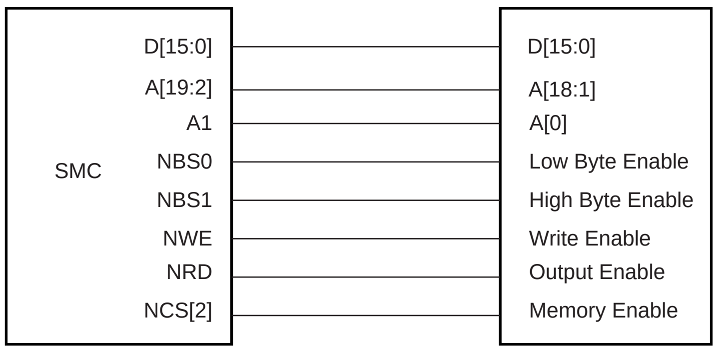
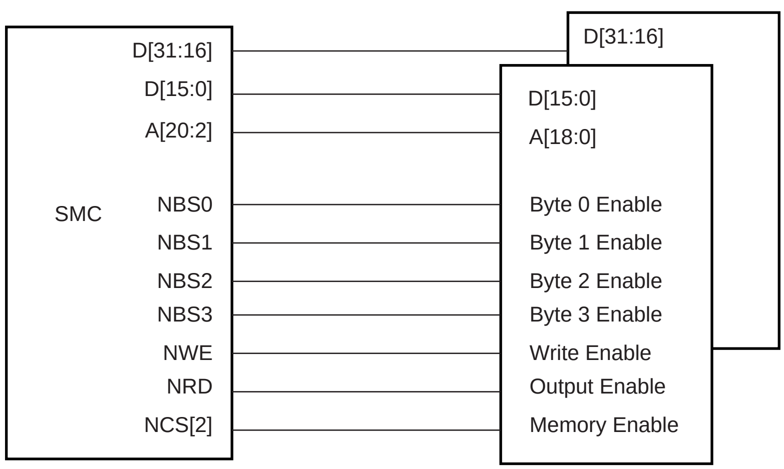


Figure 16-5. Memory Connection for a 32-bit Data Bus



16.7.2.1 Byte Write Access

Byte write access supports one byte write signal per byte of the data bus and a single read signal.

Note that the SMC does not allow boot in Byte Write Access mode.

- For 16-bit devices: the SMC provides NWR0 and NWR1 write signals for respectively byte0 (lower byte) and byte1 (upper byte) of a 16-bit bus. One single read signal (NRD) is provided.

  Byte Write Access is used to connect 2 x 8-bit devices as a 16-bit memory.

- For 32-bit devices: NWR0, NWR1, NWR2 and NWR3, are the write signals of byte0 (lower byte), byte1, byte2 and byte 3 (upper byte) respectively. One single read signal (NRD) is provided.

  Byte Write Access is used to connect 4 x 8-bit devices as a 32-bit memory.

  Byte Write option is illustrated on Figure 16-6.

16.7.2.2 Byte Select Access

In this mode, read/write operations can be enabled/disabled at a byte level. One byte-select line per byte of the data bus is provided. One NRD and one NWE signal control read and write.

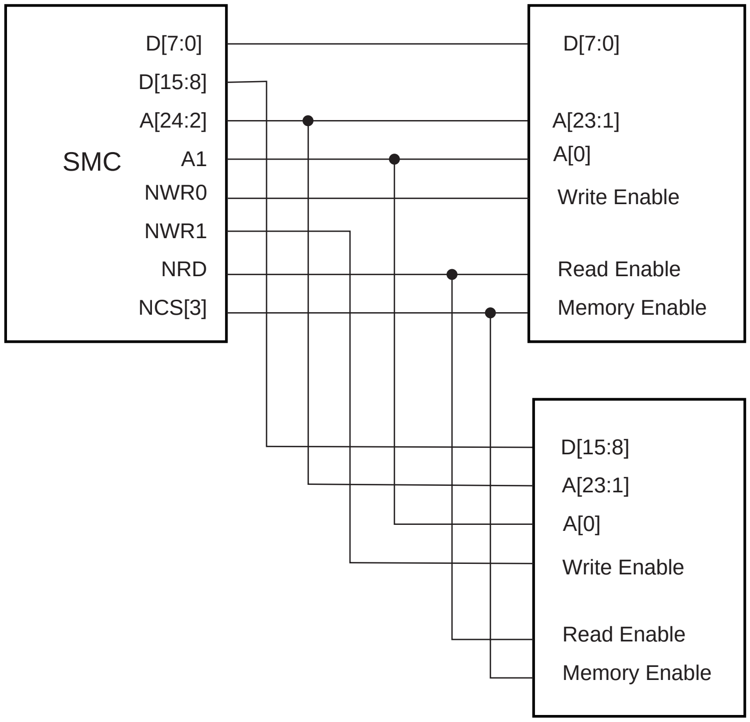
- For 16-bit devices: the SMC provides NBS0 and NBS1 selection signals for respectively byte0 (lower byte) and byte1 (upper byte) of a 16-bit bus.

  Byte Select Access is used to connect one 16-bit device.

- For 32-bit devices: NBS0, NBS1, NBS2 and NBS3, are the selection signals of byte0 (lower byte), byte1, byte2 and byte 3 (upper byte) respectively. Byte Select Access is used to connect two 16-bit devices.

Figure 16-7 shows how to connect two 16-bit devices on a 32-bit data bus in Byte Select Access mode, on NCS3 (BAT = Byte Select Access).

Figure 16-6. Connection of 2 x 8-bit Devices on a 16-bit Bus: Byte Write Option



16.7.2.3 Signal Multiplexing

Depending on the BAT, only the write signals or the byte select signals are used. To save IOs at the external bus interface, control signals at the SMC interface are multiplexed. Table 16-3 shows signal multiplexing depending on the data bus width and the byte access type.

For 32-bit devices, bits A0 and A1 are unused. For 16-bit devices, bit A0 of address is unused. When Byte Select Option is selected, NWR1 to NWR3 are unused. When Byte Write option is selected, NBS0 to NBS3 are unused.

Figure 16-7. Connection of 2x16-bit Data Bus on a 32-bit Data Bus (Byte Select Option)

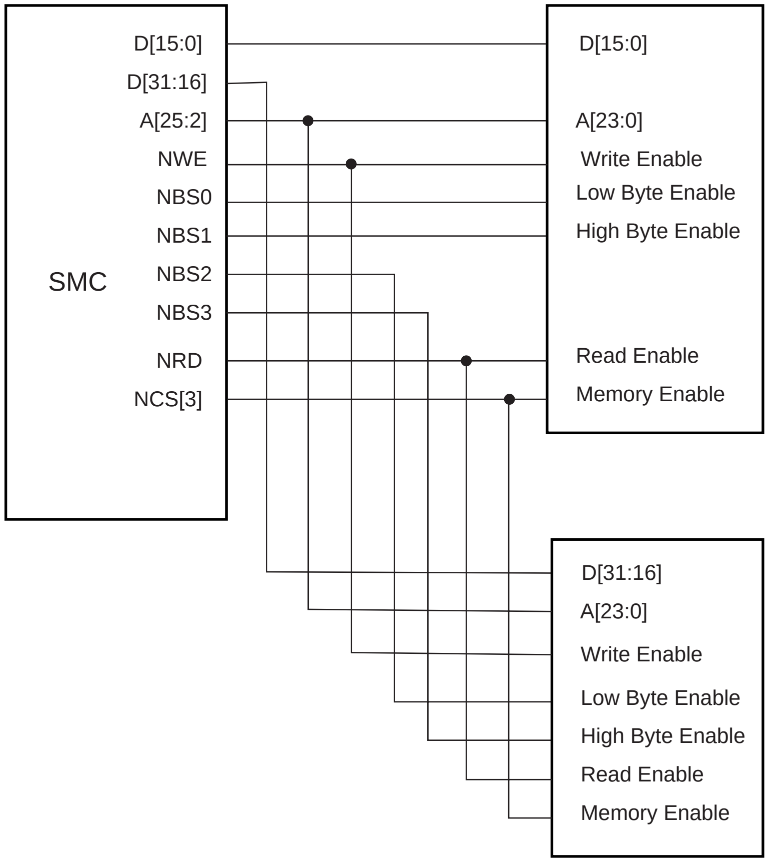


Table 16-3. SMC Multiplexed Signal Translation

Signal Name	**32-bit Bus**			**16-bit Bus**		**8-bit Bus**
Device Type	**1x32-bit**	**2x16-bit**	**4 x 8-bit**	**1x16-bit**	**2 x 8-bit**	**1 x 8-bit**
Byte Access Type (BAT)	Byte Select	Byte Select	Byte Write	Byte Select	Byte Write	
NBS0_A0	NBS0	NBS0		NBS0		A0
NWE_NWR0	NWE	NWE	NWR0	NWE	NWR0	NWE
NBS1_NWR1	NBS1	NBS1	NWR1	NBS1	NWR1	
NBS2_NWR2_A1	NBS2	NBS2	NWR2	A1	A1	A1
NBS3_NWR3	NBS3	NBS3	NWR3			

16.8 Standard Read and Write Protocols

In the following sections, the byte access type is not considered. Byte select lines (NBS0 to NBS3) always have the same timing as the A address bus. NWE represents either the NWE signal in byte select access type or one of the byte write lines (NWR0 to NWR3) in byte write access type. NWR0 to NWR3 have the same timings and protocol as NWE. In the same way, NCS represents one of the NCS[0..7] chip select lines.

16.8.1 Read Waveforms

The read cycle is shown on Figure 16-8.

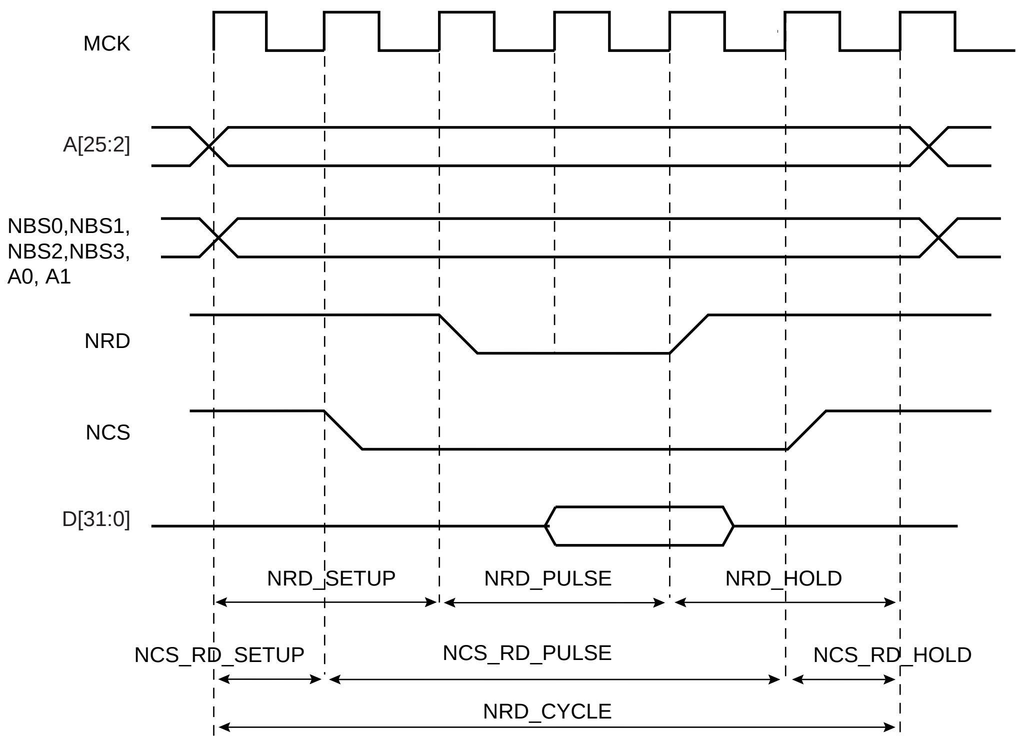
The read cycle starts with the address setting on the memory address bus, i.e.:

{A[25:2], A1, A0} for 8-bit devices

{A[25:2], A1} for 16-bit devices

A[25:2] for 32-bit devices.

Figure 16-8. Standard Read Cycle



16.8.1.1 NRD Waveform

The NRD signal is characterized by a setup timing, a pulse width and a hold timing.

1. NRD_SETUP: the NRD setup time is defined as the setup of address before the NRD falling edge;
2. NRD_PULSE: the NRD pulse length is the time between NRD falling edge and NRD rising edge;
3. NRD_HOLD: the NRD hold time is defined as the hold time of address after the NRD rising edge.

16.8.1.2 NCS Waveform

Similarly, the NCS signal can be divided into a setup time, pulse length and hold time:

1. NCS_RD_SETUP: the NCS setup time is defined as the setup time of address before the NCS falling edge.
2. NCS_RD_PULSE: the NCS pulse length is the time between NCS falling edge and NCS rising edge;
3. NCS_RD_HOLD: the NCS hold time is defined as the hold time of address after the NCS rising edge.

16.8.1.3 Read Cycle

The NRD_CYCLE time is defined as the total duration of the read cycle, i.e., from the time where address is set on the address bus to the point where address may change. The total read cycle time is equal to:

NRD_CYCLE = NRD_SETUP + NRD_PULSE + NRD_HOLD

= NCS_RD_SETUP + NCS_RD_PULSE + NCS_RD_HOLD

All NRD and NCS timings are defined separately for each chip select as an integer number of Master Clock cycles. To ensure that the NRD and NCS timings are coherent, user must define the total read cycle instead of the hold timing. NRD_CYCLE implicitly defines the NRD hold time and NCS hold time as:

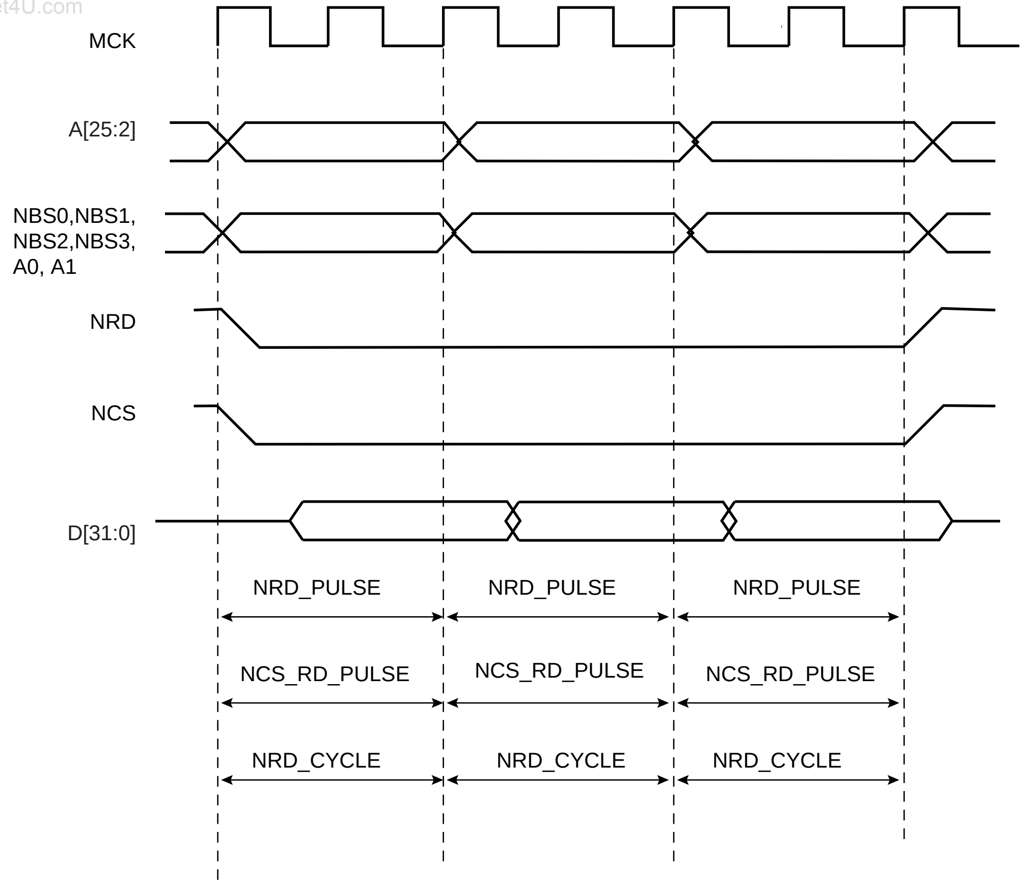
NRD_HOLD = NRD_CYCLE - NRD SETUP - NRD PULSE

NCS_RD_HOLD = NRD_CYCLE - NCS_RD_SETUP - NCS_RD_PULSE

16.8.1.4 Null Delay Setup and Hold

If null setup and hold parameters are programmed for NRD and/or NCS, NRD and NCS remain active continuously in case of consecutive read cycles in the same memory (see Figure 16-9).

Figure 16-9. No Setup, No Hold On NRD and NCS Read Signals



16.8.1.5 Null Pulse

Programming null pulse is not permitted. Pulse must be at least set to 1. A null value leads to unpredictable behavior.

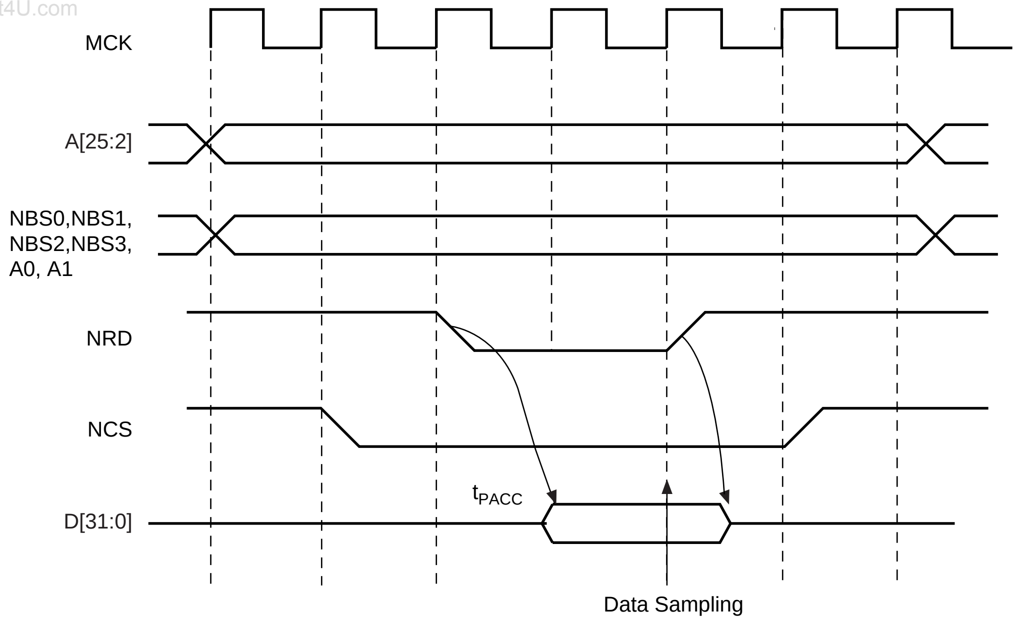
16.8.2 Read Mode

As NCS and NRD waveforms are defined independently of one other, the SMC needs to know when the read data is available on the data bus. The SMC does not compare NCS and NRD timings to know which signal rises first. The READ_MODE parameter in the SMC_MODE register of the corresponding chip select indicates which signal of NRD and NCS controls the read operation.

16.8.2.1 Read is Controlled by NRD (READ_MODE = 1):

Figure 16-10 shows the waveforms of a read operation of a typical asynchronous RAM. The read data is available t_{PACC} after the falling edge of NRD, and turns to 'Z' after the rising edge of NRD. In this case, the READ_MODE must be set to 1 (read is controlled by NRD), to indicate that data is available with the rising edge of NRD. The SMC samples the read data internally on the rising edge of Master Clock that generates the rising edge of NRD, whatever the programmed waveform of NCS may be.

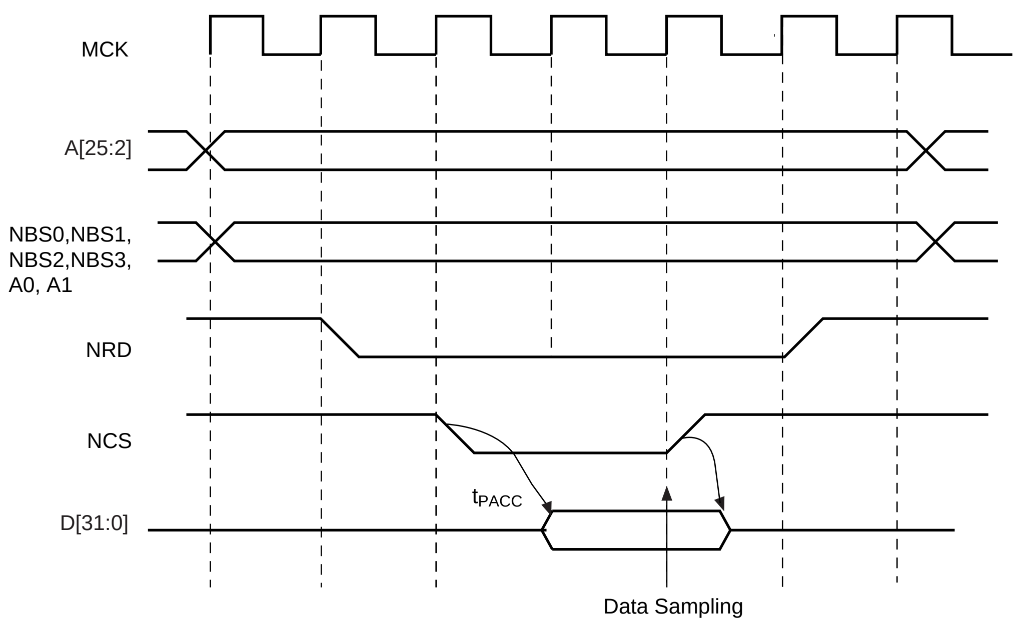
Figure 16-10. READ_MODE = 1: Data is sampled by SMC before the rising edge of NRD



16.8.2.2 *Read is Controlled by NCS (READ_MODE = 0)*

Figure 16-11 shows the typical read cycle of an LCD module. The read data is valid t_{PACC} after the falling edge of the NCS signal and remains valid until the rising edge of NCS. Data must be sampled when NCS is raised. In that case, the READ_MODE must be set to 0 (read is controlled by NCS): the SMC internally samples the data on the rising edge of Master Clock that generates the rising edge of NCS, whatever the programmed waveform of NRD may be.

Figure 16-11. READ_MODE = 0: Data is sampled by SMC before the rising edge of NCS



16.8.3 Write Waveforms

The write protocol is similar to the read protocol. It is depicted in Figure 16-12. The write cycle starts with the address setting on the memory address bus.

16.8.3.1 NWE Waveforms

The NWE signal is characterized by a setup timing, a pulse width and a hold timing.

1. NWE_SETUP: the NWE setup time is defined as the setup of address and data before the NWE falling edge;
2. NWE_PULSE: The NWE pulse length is the time between NWE falling edge and NWE rising edge;
3. NWE_HOLD: The NWE hold time is defined as the hold time of address and data after the NWE rising edge.

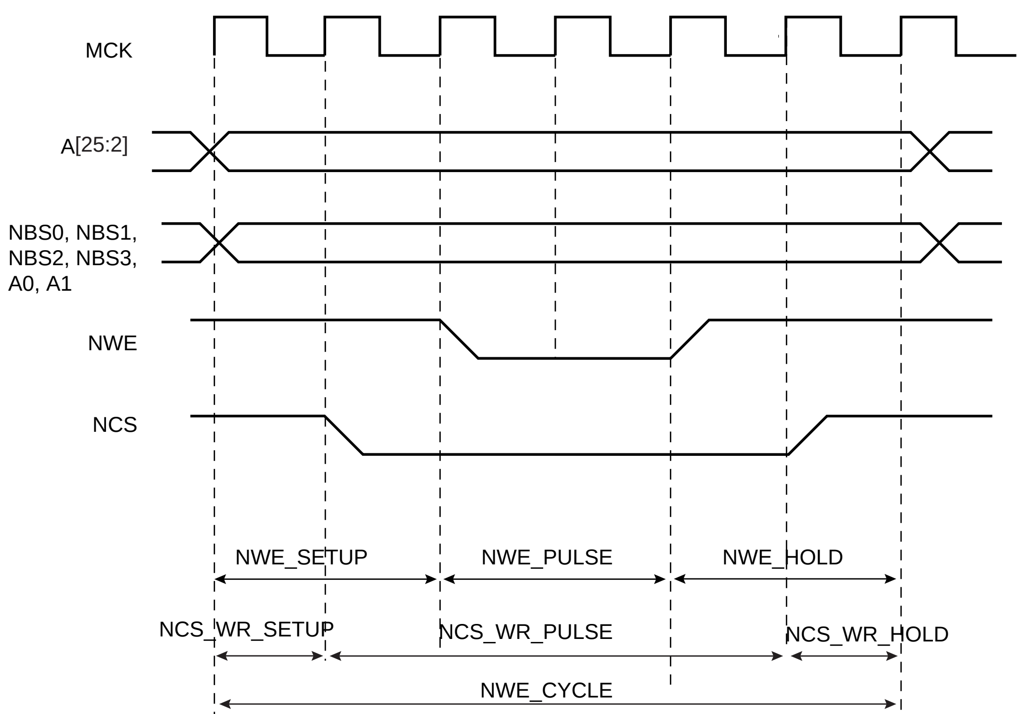
The NWE waveforms apply to all byte-write lines in Byte Write access mode: NWR0 to NWR3.

16.8.3.2 NCS Waveforms

The NCS signal waveforms in write operation are not the same that those applied in read operations, but are separately defined:

1. NCS_WR_SETUP: the NCS setup time is defined as the setup time of address before the NCS falling edge.
2. NCS_WR_PULSE: the NCS pulse length is the time between NCS falling edge and NCS rising edge;
3. NCS_WR_HOLD: the NCS hold time is defined as the hold time of address after the NCS rising edge.

Figure 16-12. Write Cycle



16.8.3.3 Write Cycle

The write_cycle time is defined as the total duration of the write cycle, that is, from the time where address is set on the address bus to the point where address may change. The total write cycle time is equal to:

NWE_CYCLE = NWE_SETUP + NWE_PULSE + NWE_HOLD

= NCS_WR_SETUP + NCS_WR_PULSE + NCS_WR_HOLD

All NWE and NCS (write) timings are defined separately for each chip select as an integer number of Master Clock cycles. To ensure that the NWE and NCS timings are coherent, the user must define the total write cycle instead of the hold timing. This implicitly defines the NWE hold time and NCS (write) hold times as:

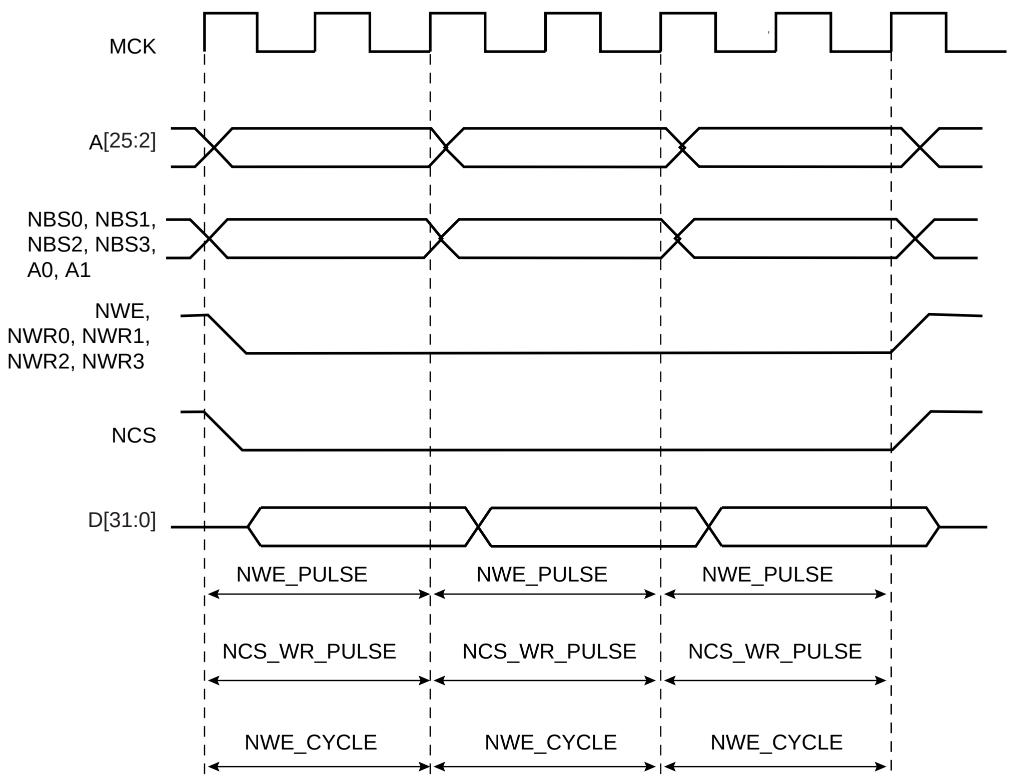
NWE_HOLD = NWE_CYCLE - NWE_SETUP - NWE_PULSE

NCS_WR_HOLD = NWE_CYCLE - NCS_WR_SETUP - NCS_WR_PULSE

16.8.3.4 Null Delay Setup and Hold

If null setup parameters are programmed for NWE and/or NCS, NWE and/or NCS remain active continuously in case of consecutive write cycles in the same memory (see Figure 16-13). However, for devices that perform write operations on the rising edge of NWE or NCS, such as SRAM, either a setup or a hold must be programmed.

Figure 16-13. Null Setup and Hold Values of NCS and NWE in Write Cycle



16.8.3.5 Null Pulse

Programming null pulse is not permitted. Pulse must be at least set to 1. A null value leads to unpredictable behavior.

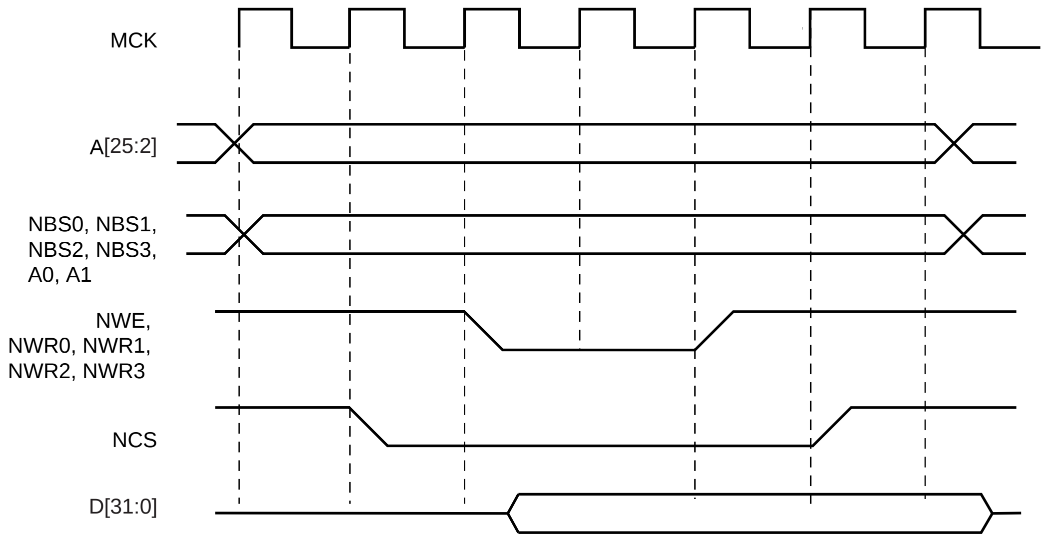
16.8.4 Write Mode

The WRITE_MODE parameter in the SMC_MODE register of the corresponding chip select indicates which signal controls the write operation.

16.8.4.1 Write is Controlled by NWE (WRITE_MODE = 1):

Figure 16-14 shows the waveforms of a write operation with WRITE_MODE set to 1. The data is put on the bus during the pulse and hold steps of the NWE signal. The internal data buffers are turned out after the NWE_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NCS.

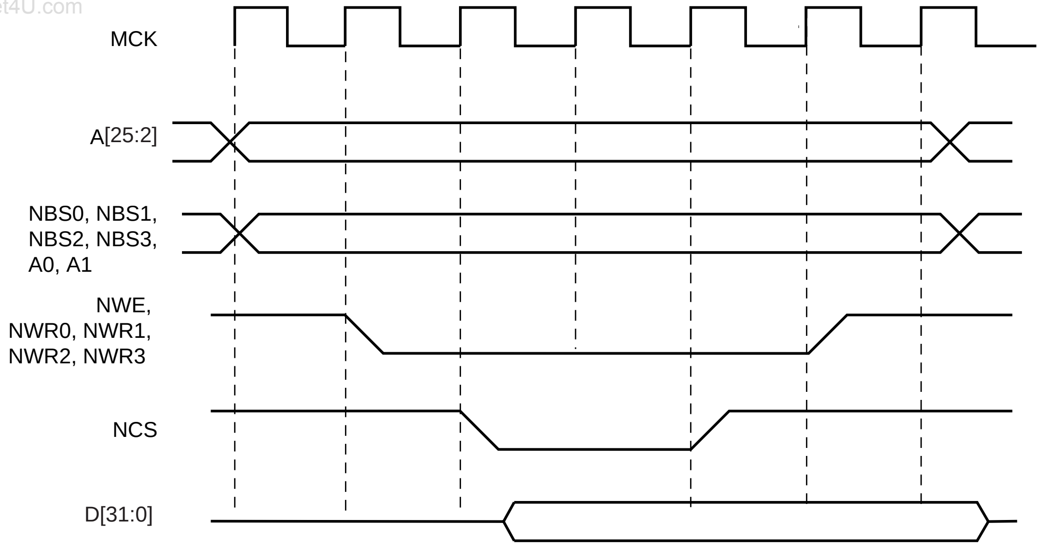
Figure 16-14. WRITE_MODE = 1. The write operation is controlled by NWE



16.8.4.2 Write is Controlled by NCS (WRITE_MODE = 0)

Figure 16-15 shows the waveforms of a write operation with WRITE_MODE set to 0. The data is put on the bus during the pulse and hold steps of the NCS signal. The internal data buffers are turned out after the NCS_WR_SETUP time, and until the end of the write cycle, regardless of the programmed waveform on NWE.

Figure 16-15. WRITE_MODE = 0. The write operation is controlled by NCS



16.8.5 Coding Timing Parameters

All timing parameters are defined for one chip select and are grouped together in one SMC_REGISTER according to their type.

The SMC_SETUP register groups the definition of all setup parameters:

• NRD_SETUP, NCS_RD_SETUP, NWE_SETUP, NCS_WR_SETUP

The SMC_PULSE register groups the definition of all pulse parameters:

• NRD_PULSE, NCS_RD_PULSE, NWE_PULSE, NCS_WR_PULSE

The SMC_CYCLE register groups the definition of all cycle parameters:

• NRD_CYCLE, NWE_CYCLE

Table 16-4 shows how the timing parameters are coded and their permitted range.

Table 16-4. Coding and Range of Timing Parameters

Coded Value	Number of Bits	Effective Value	Permitted Range	
			Coded Value	Effective Value
setup [5:0]	6	128 x setup[5] + setup[4:0]	0 ≤ ≤ 31	128 ≤ ≤ 128+31
pulse [6:0]	7	256 x pulse[6] + pulse[5:0]	0 ≤ ≤ 63	256 ≤ ≤ 256+63
cycle [8:0]	9	256 x cycle[8:7] + cycle[6:0]	0 ≤ ≤ 127	256 ≤ ≤ 256+127 512 ≤ ≤ 512+127 768 ≤ ≤ 768+127

16.8.6 Reset Values of Timing Parameters

Table 16-5 gives the default value of timing parameters at reset.

Table 16-5. Reset Values of Timing Parameters

Register	Reset Value	
SMC_SETUP	0x00000000	All setup timings are set to 1
SMC_PULSE	0x01010101	All pulse timings are set to 1
SMC_CYCLE	0x00010001	The read and write operation last 3 Master Clock cycles and provide one hold cycle
WRITE_MODE	1	Write is controlled with NWE
READ_MODE	1	Read is controlled with NRD

16.8.7 Usage Restriction

The SMC does not check the validity of the user-programmed parameters. If the sum of SETUP and PULSE parameters is larger than the corresponding CYCLE parameter, this leads to unpredictable behavior of the SMC.

For read operations:

Null but positive setup and hold of address and NRD and/or NCS can not be guaranteed at the memory interface because of the propagation delay of theses signals through external logic and pads. If positive setup and hold values must be verified, then it is strictly recommended to program non-null values so as to cover possible skews between address, NCS and NRD signals.

For write operations:

If a null hold value is programmed on NWE, the SMC can guarantee a positive hold of address, byte select lines, and NCS signal after the rising edge of NWE. This is true for WRITE_MODE = 1 only. See "Early Read Wait State" on page 169.

For read and write operations: a null value for pulse parameters is forbidden and may lead to unpredictable behavior.

In read and write cycles, the setup and hold time parameters are defined in reference to the address bus. For external devices that require setup and hold time between NCS and NRD signals (read), or between NCS and NWE signals (write), these setup and hold times must be converted into setup and hold times in reference to the address bus.

16.9 Automatic Wait States

Under certain circumstances, the SMC automatically inserts idle cycles between accesses to avoid bus contention or operation conflict.

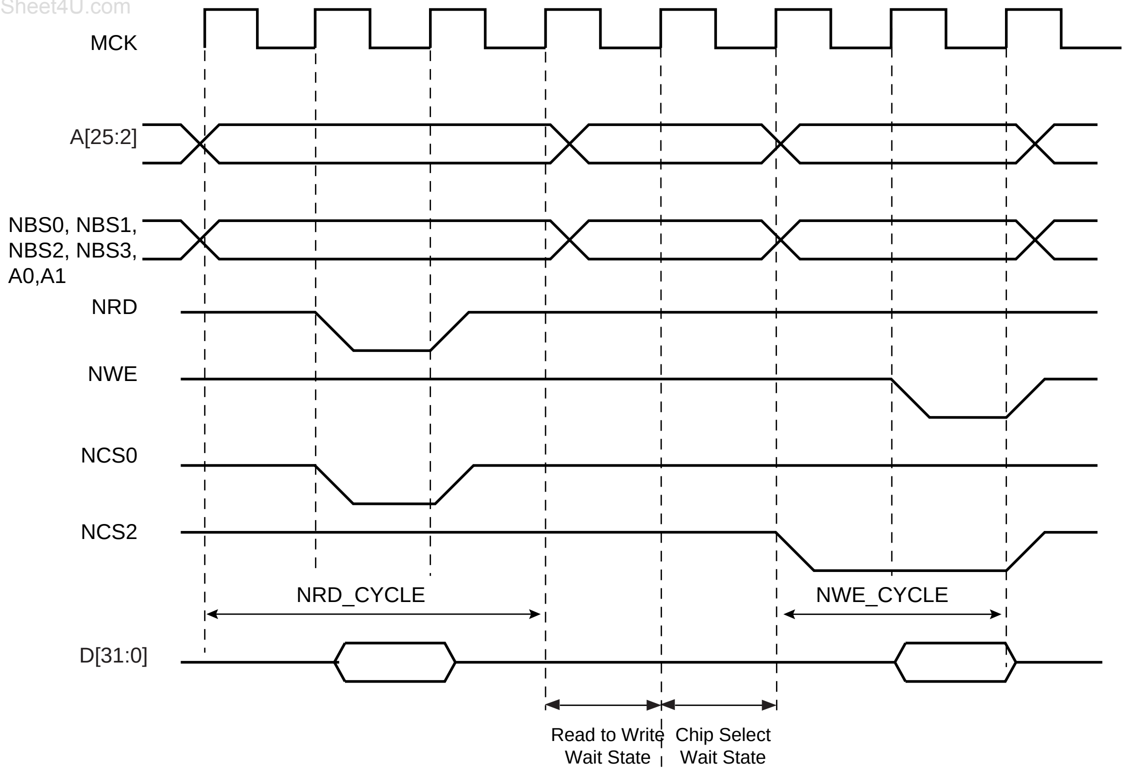
16.9.1 Chip Select Wait States

The SMC always inserts an idle cycle between 2 transfers on separate chip selects. This idle cycle ensures that there is no bus contention between the de-activation of one device and the activation of the next one.

During chip select wait state, all control lines are turned inactive: NBS0 to NBS3, NWR0 to NWR3, NCS[0..7], NRD lines are all set to 1.

Figure 16-16 illustrates a chip select wait state between access on Chip Select 0 and Chip Select 2.

Figure 16-16. Chip Select Wait State between a Read Access on NCS0 and a Write Access on NCS2



16.9.2 Early Read Wait State

In some cases, the SMC inserts a wait state cycle between a write access and a read access to allow time for the write cycle to end before the subsequent read cycle begins. This wait state is not generated in addition to a chip select wait state. The early read cycle thus only occurs between a write and read access to the same memory device (same chip select).

An early read wait state is automatically inserted if at least one of the following conditions is valid:

- if the write controlling signal has no hold time and the read controlling signal has no setup time (Figure 16-17).
- in NCS write controlled mode (WRITE_MODE = 0), if there is no hold timing on the NCS signal and the NCS_RD_SETUP parameter is set to 0, regardless of the read mode (Figure 16-18). The write operation must end with a NCS rising edge. Without an Early Read Wait State, the write operation could not complete properly.
- in NWE controlled mode (WRITE_MODE = 1) and if there is no hold timing (NWE_HOLD = 0), the feedback of the write control signal is used to control address, data, chip select and byte select lines. If the external write control signal is not inactivated as expected due to load capacitances, an Early Read Wait State is inserted and address, data and control signals are maintained one more cycle. See Figure 16-19.

Figure 16-17. Early Read Wait State: Write with No Hold Followed by Read with No Setup

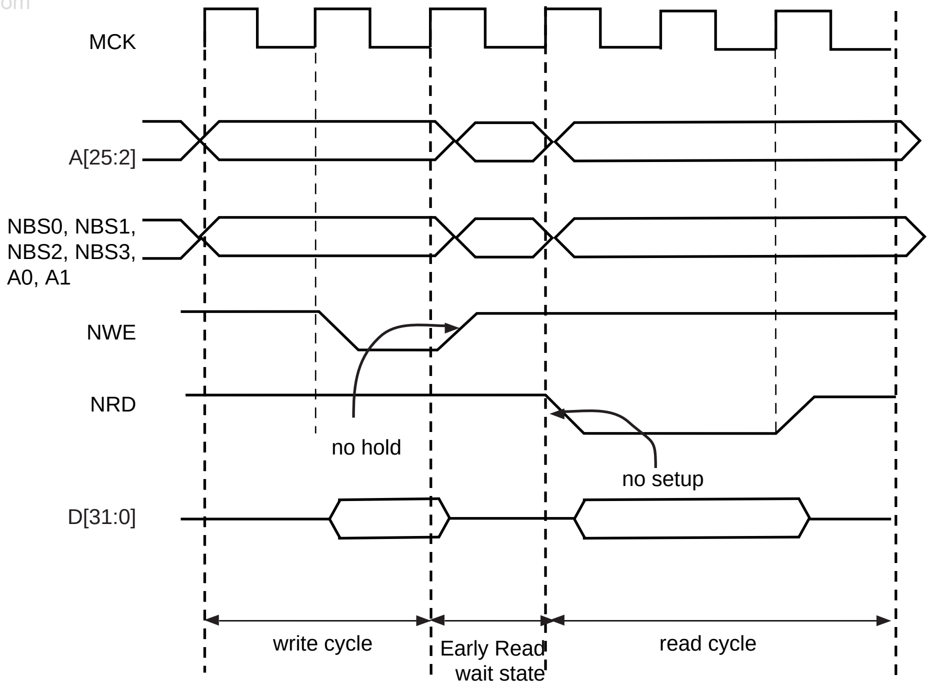


Figure 16-18. Early Read Wait State: NCS Controlled Write with No Hold Followed by a Read with No NCS Setup

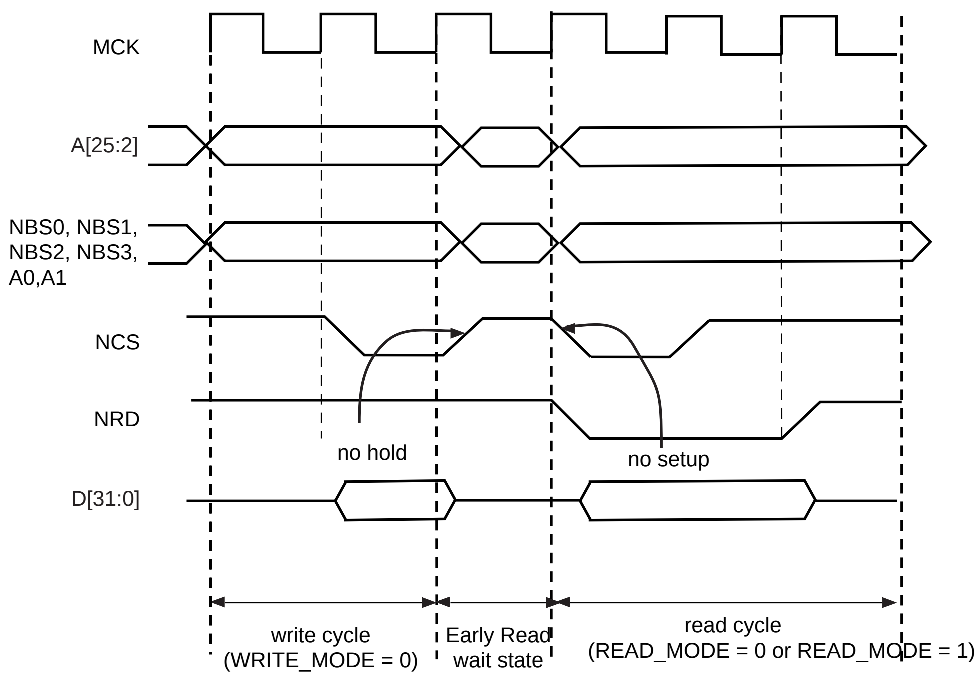
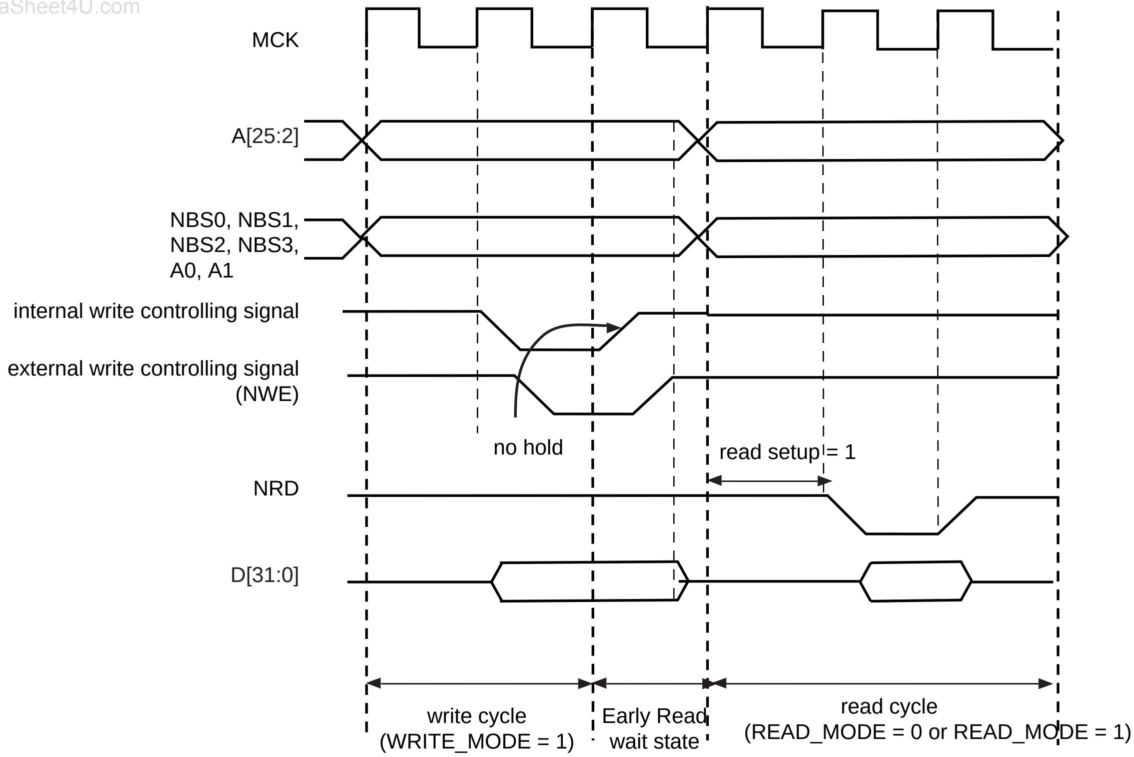


Figure 16-19. Early Read Wait State: NWE-controlled Write with No Hold Followed by a Read with one Set-up Cycle



16.9.3 Reload User Configuration Wait State

The user may change any of the configuration parameters by writing the SMC user interface.

When detecting that a new user configuration has been written in the user interface, the SMC inserts a wait state before starting the next access. The so called “Reload User Configuration Wait State” is used by the SMC to load the new set of parameters to apply to next accesses.

The Reload Configuration Wait State is not applied in addition to the Chip Select Wait State. If accesses before and after re-programming the user interface are made to different devices (Chip Selects), then one single Chip Select Wait State is applied.

On the other hand, if accesses before and after writing the user interface are made to the same device, a Reload Configuration Wait State is inserted, even if the change does not concern the current Chip Select.

16.9.3.1 User Procedure

To insert a Reload Configuration Wait State, the SMC detects a write access to any SMC_MODE register of the user interface. If the user only modifies timing registers (SMC_SETUP, SMC_PULSE, SMC_CYCLE registers) in the user interface, he must validate the modification by writing the SMC_MODE, even if no change was made on the mode parameters.

16.9.3.2 Slow Clock Mode Transition

A Reload Configuration Wait State is also inserted when the Slow Clock Mode is entered or exited, after the end of the current transfer (see “Slow Clock Mode” on page 183).

16.9.4 Read to Write Wait State

Due to an internal mechanism, a wait cycle is always inserted between consecutive read and write SMC accesses.

This wait cycle is referred to as a read to write wait state in this document.

This wait cycle is applied in addition to chip select and reload user configuration wait states when they are to be inserted. See Figure 16-16 on page 169.

16.10 Data Float Wait States

Some memory devices are slow to release the external bus. For such devices, it is necessary to add wait states (data float wait states) after a read access:

- before starting a read access to a different external memory
- before starting a write access to the same device or to a different external one.

The Data Float Output Time (t_{DF}) for each external memory device is programmed in the TDF_CYCLES field of the SMC_MODE register for the corresponding chip select. The value of TDF_CYCLES indicates the number of data float wait cycles (between 0 and 15) before the external device releases the bus, and represents the time allowed for the data output to go to high impedance after the memory is disabled.

Data float wait states do not delay internal memory accesses. Hence, a single access to an external memory with long t_{DF} will not slow down the execution of a program from internal memory.

The data float wait states management depends on the READ_MODE and the TDF_MODE fields of the SMC_MODE register for the corresponding chip select.

16.10.1 READ_MODE

Setting the READ_MODE to 1 indicates to the SMC that the NRD signal is responsible for turning off the tri-state buffers of the external memory device. The Data Float Period then begins after the rising edge of the NRD signal and lasts TDF_CYCLES MCK cycles.

When the read operation is controlled by the NCS signal (READ_MODE = 0), the TDF field gives the number of MCK cycles during which the data bus remains busy after the rising edge of NCS.

Figure 16-20 illustrates the Data Float Period in NRD-controlled mode (READ_MODE =1), assuming a data float period of 2 cycles (TDF_CYCLES = 2). Figure 16-21 shows the read operation when controlled by NCS (READ_MODE = 0) and the TDF_CYCLES parameter equals 3.

Figure 16-20. TDF Period in NRD Controlled Read Access (TDF = 2)

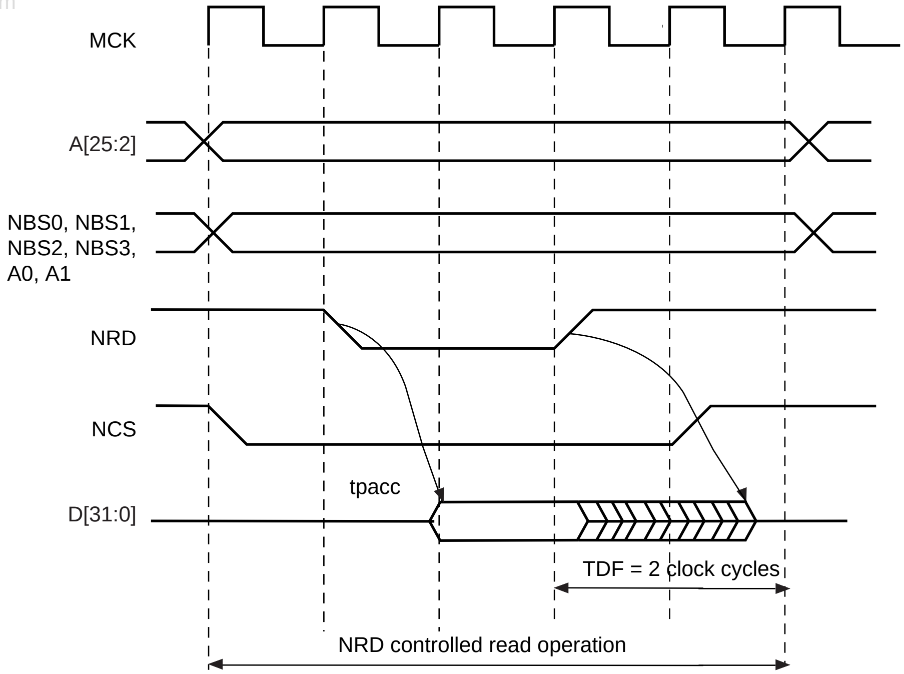
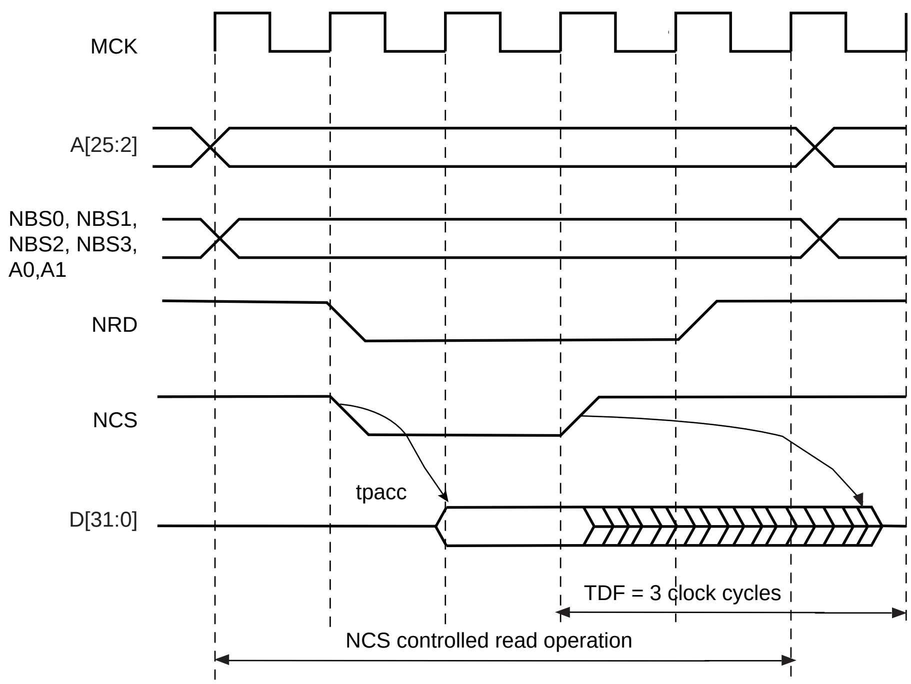


Figure 16-21. TDF Period in NCS Controlled Read Operation (TDF = 3)



16.10.2 TDF Optimization Enabled (TDF_MODE = 1)

When the TDF_MODE of the SMC_MODE register is set to 1 (TDF optimization is enabled), the SMC takes advantage of the setup period of the next access to optimize the number of wait states cycle to insert.

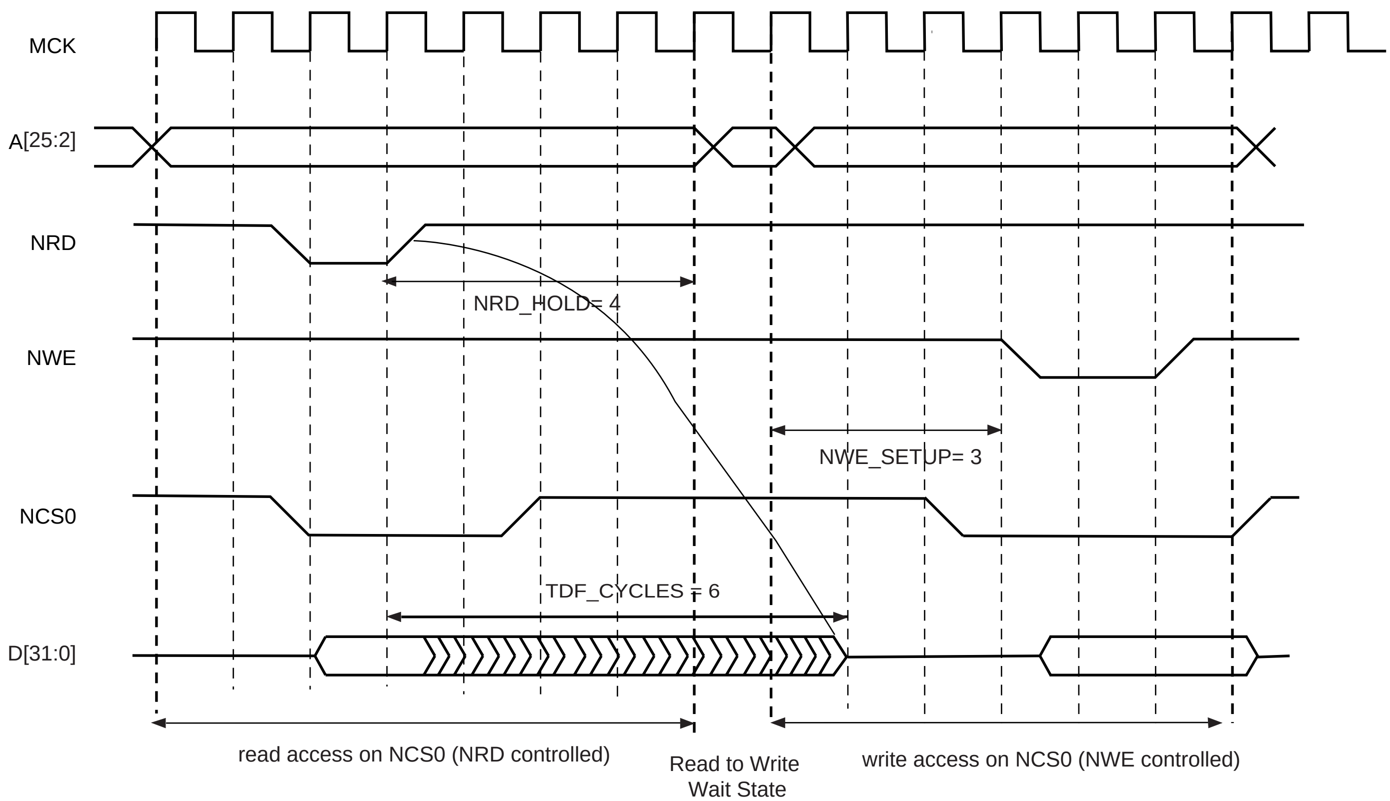
Figure 16-22 shows a read access controlled by NRD, followed by a write access controlled by NWE, on Chip Select 0. Chip Select 0 has been programmed with:

NRD_HOLD = 4; READ_MODE = 1 (NRD controlled)

NWE_SETUP = 3; WRITE_MODE = 1 (NWE controlled)

TDF_CYCLES = 6; TDF_MODE = 1 (optimization enabled).

Figure 16-22. TDF Optimization: No TDF wait states are inserted if the TDF period is over when the next access begins



16.10.3 TDF Optimization Disabled (TDF_MODE = 0)

When optimization is disabled, tdf wait states are inserted at the end of the read transfer, so that the data float period is ended when the second access begins. If the hold period of the read1 controlling signal overlaps the data float period, no additional tdf wait states will be inserted.

Figure 16-23, Figure 16-24 and Figure 16-25 illustrate the cases:

- read access followed by a read access on another chip select,
- read access followed by a write access on another chip select,
- read access followed by a write access on the same chip select,

with no TDF optimization.

Figure 16-23. TDF Optimization Disabled (TDF Mode = 0). TDF wait states between 2 read accesses on different chip selects

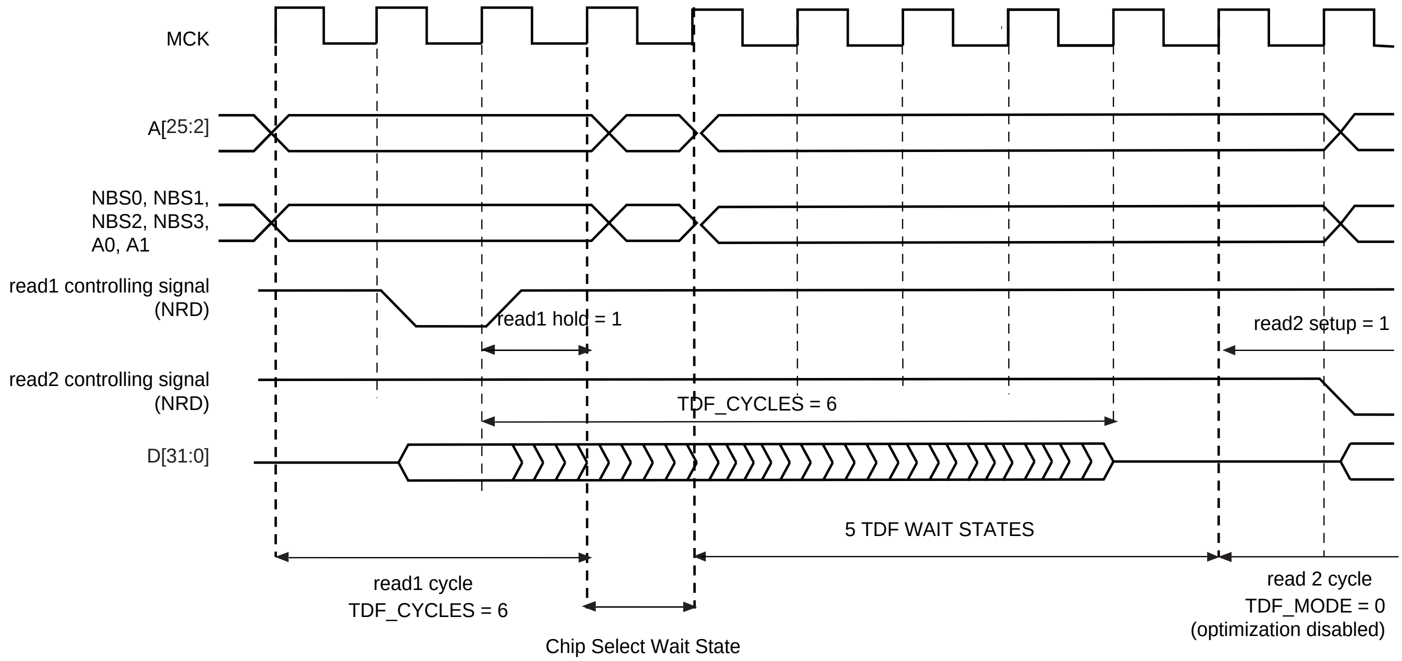


Figure 16-24. TDF Mode = 0: TDF wait states between a read and a write access on different chip selects

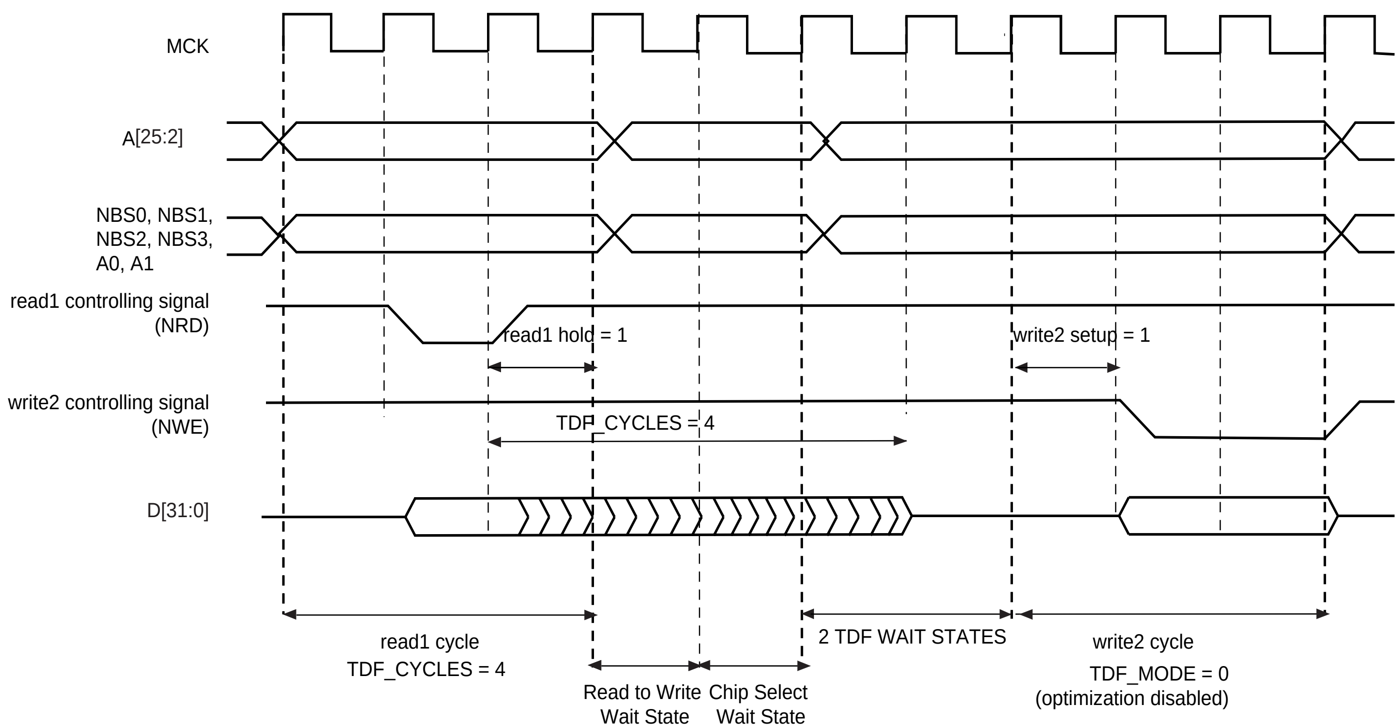
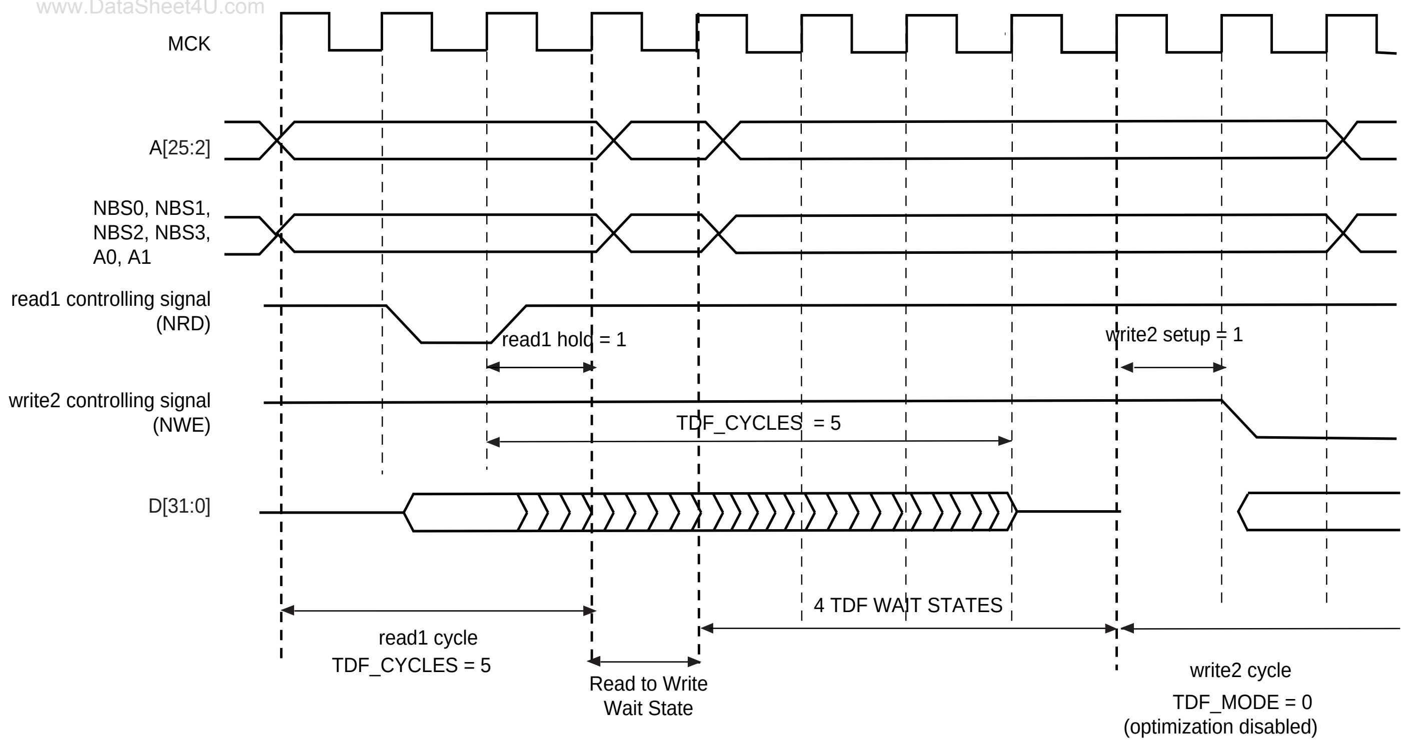


Figure 16-25. TDF Mode = 0: TDF wait states between read and write accesses on the same chip select



16.11 External Wait

Any access can be extended by an external device using the NWAIT input signal of the SMC. The EXNW_MODE field of the SMC_MODE register on the corresponding chip select must be set to either "10" (frozen mode) or "11" (ready mode). When the EXNW_MODE is set to "00" (disabled), the NWAIT signal is simply ignored on the corresponding chip select. The NWAIT signal delays the read or write operation in regards to the read or write controlling signal, depending on the read and write modes of the corresponding chip select.

16.11.1 Restriction

When one of the EXNW_MODE is enabled, **it is mandatory to program at least one hold cycle for the read/write controlling signal. For that reason, the NWAIT signal cannot be used in Page Mode ("Asynchronous Page Mode" on page 186), or in Slow Clock Mode ("Slow Clock Mode" on page 183).**

The NWAIT signal is assumed to be a response of the external device to the read/write request of the SMC. Then NWAIT is examined by the SMC only in the pulse state of the read or write controlling signal. The assertion of the NWAIT signal outside the expected period has no impact on SMC behavior.

16.11.2 Frozen Mode

When the external device asserts the NWAIT signal (active low), and after internal synchronization of this signal, the SMC state is frozen, i.e., SMC internal counters are frozen, and all control signals remain unchanged. When the resynchronized NWAIT signal is deasserted, the SMC completes the access, resuming the access from the point where it was stopped. See Figure 16-26. This mode must be selected when the external device uses the NWAIT signal to delay the access and to freeze the SMC.

The assertion of the NWAIT signal outside the expected period is ignored as illustrated in Figure 16-27.

Figure 16-26. Write Access with NWAIT Assertion in Frozen Mode (EXNW_MODE = 10)

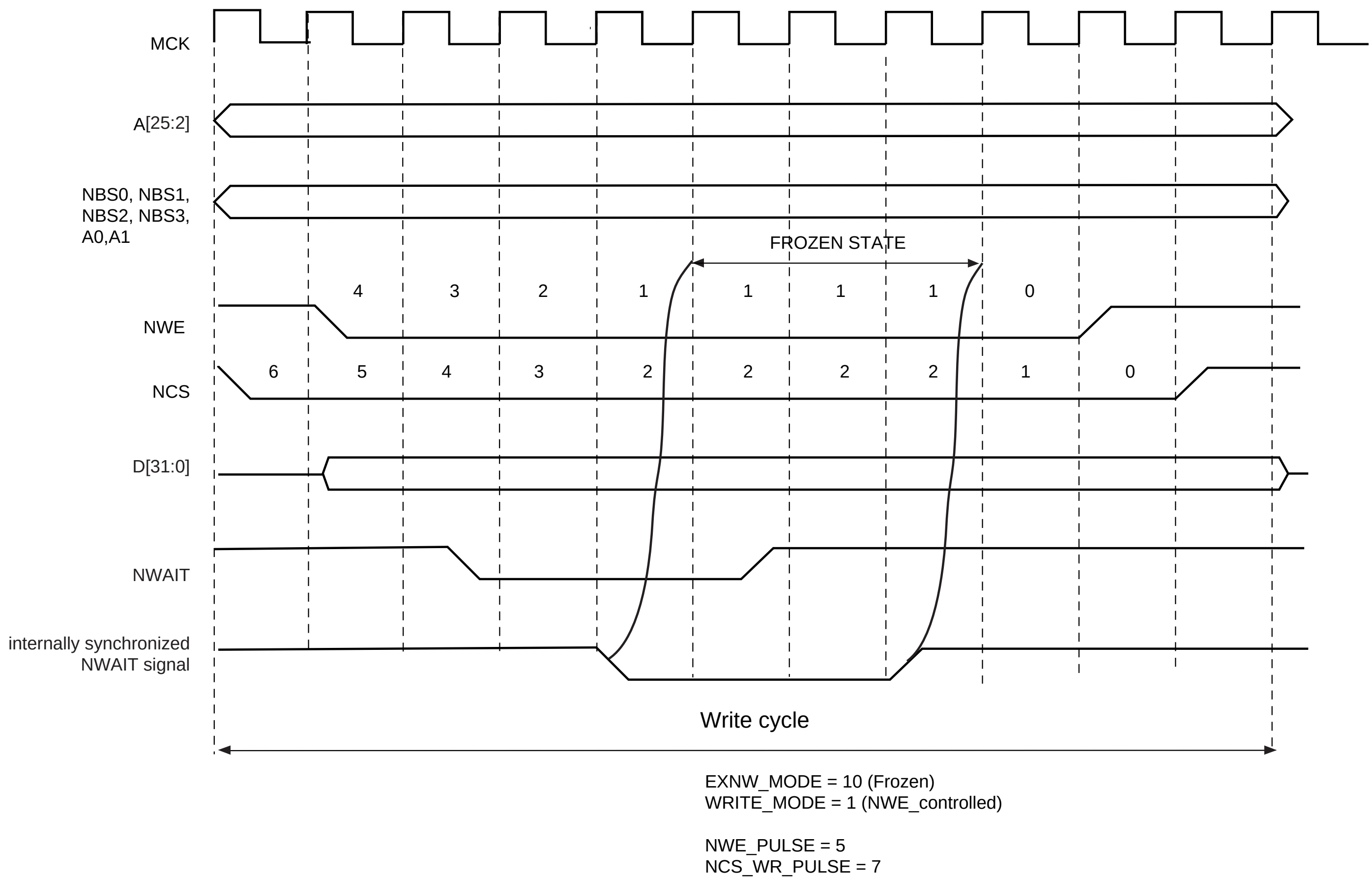
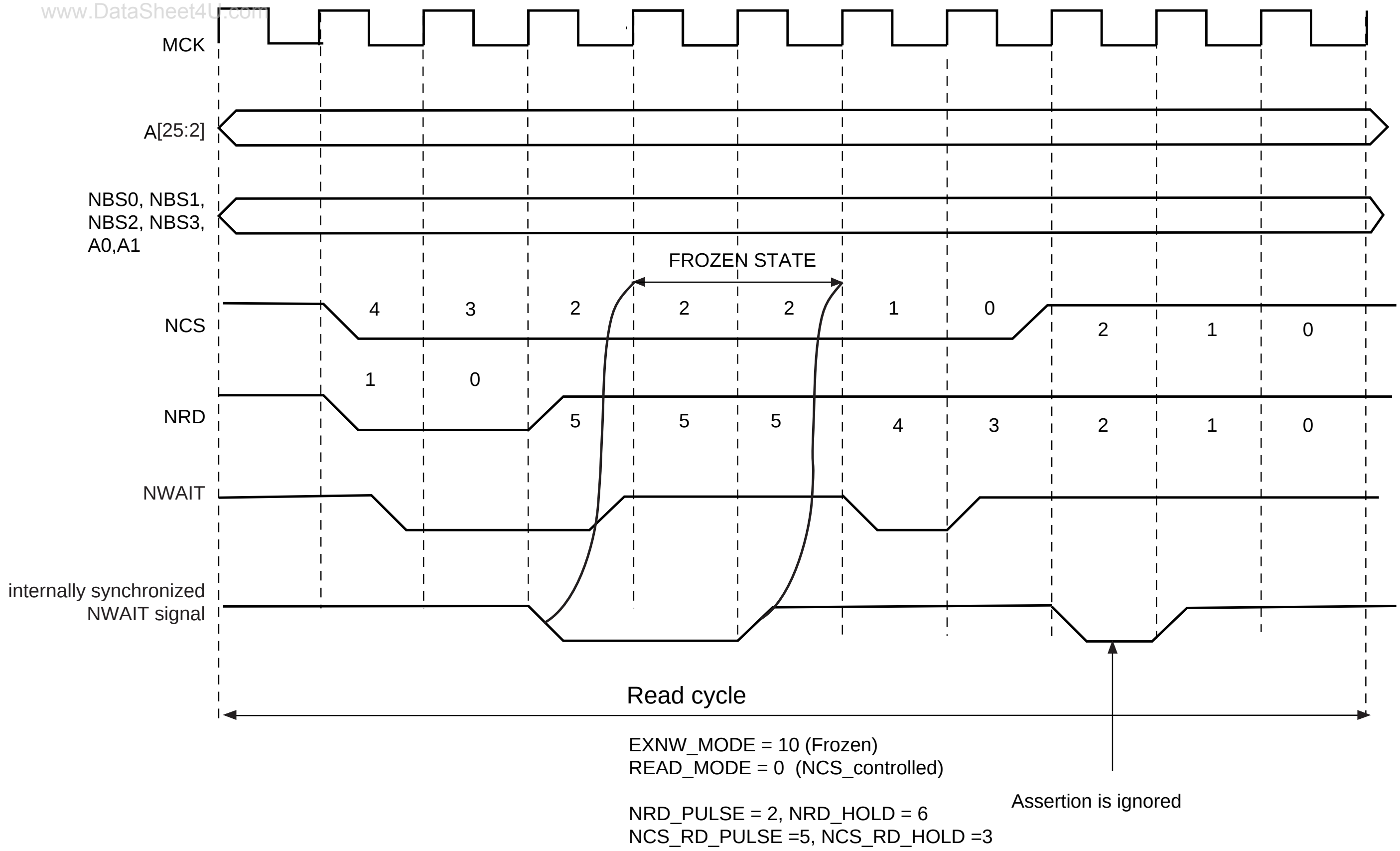


Figure 16-27. Read Access with NWAIT Assertion in Frozen Mode (EXNW_MODE = 10)



16.11.3 Ready Mode

In Ready mode (EXNW_MODE = 11), the SMC behaves differently. Normally, the SMC begins the access by down counting the setup and pulse counters of the read/write controlling signal. In the last cycle of the pulse phase, the resynchronized NWAIT signal is examined.

If asserted, the SMC suspends the access as shown in Figure 16-28 and Figure 16-29. After deassertion, the access is completed: the hold step of the access is performed.

This mode must be selected when the external device uses deassertion of the NWAIT signal to indicate its ability to complete the read or write operation.

If the NWAIT signal is deasserted before the end of the pulse, or asserted after the end of the pulse of the controlling read/write signal, it has no impact on the access length as shown in Figure 16-29.

Figure 16-28. NWAIT Assertion in Write Access: Ready Mode (EXNW_MODE = 11)

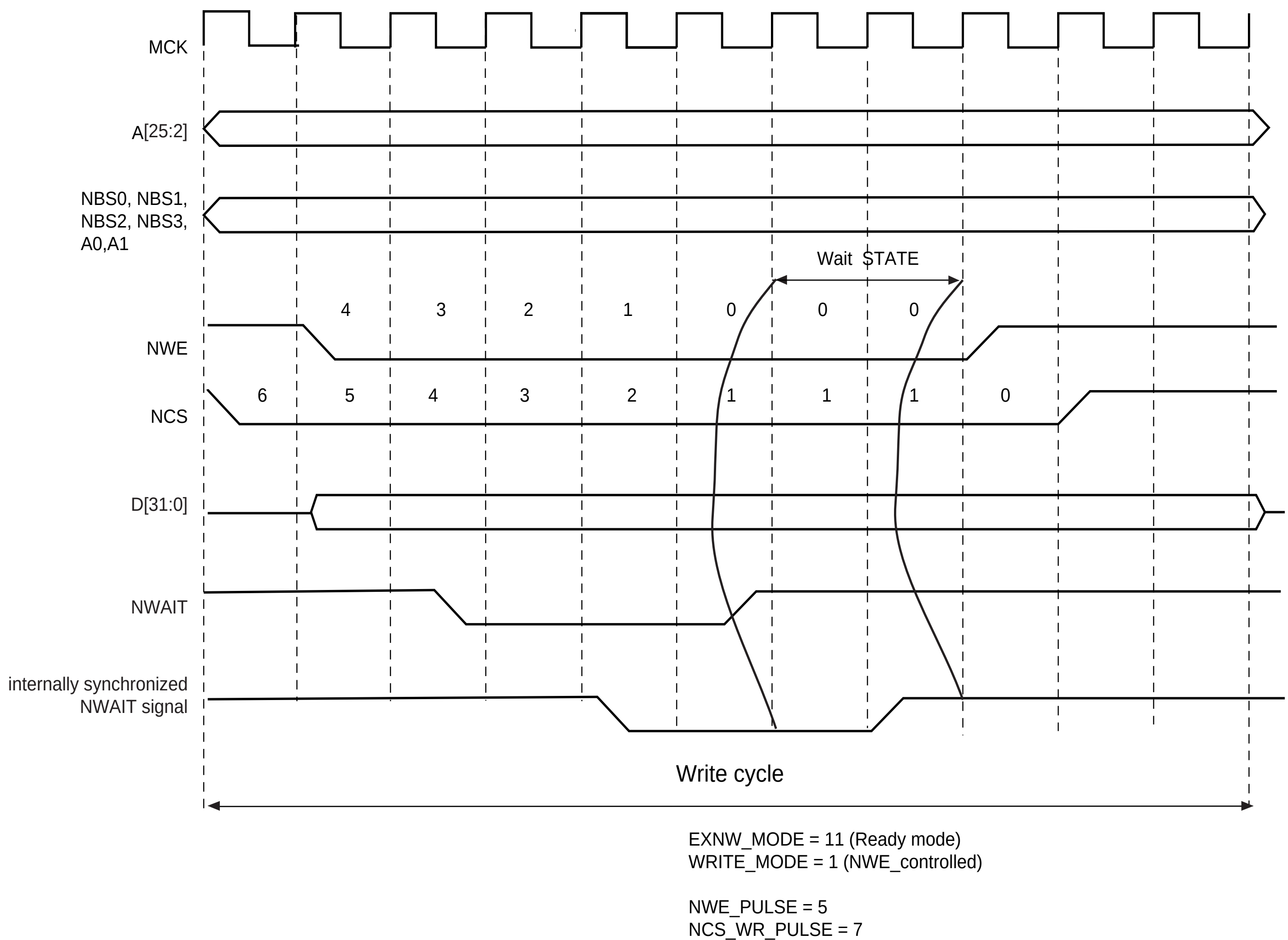
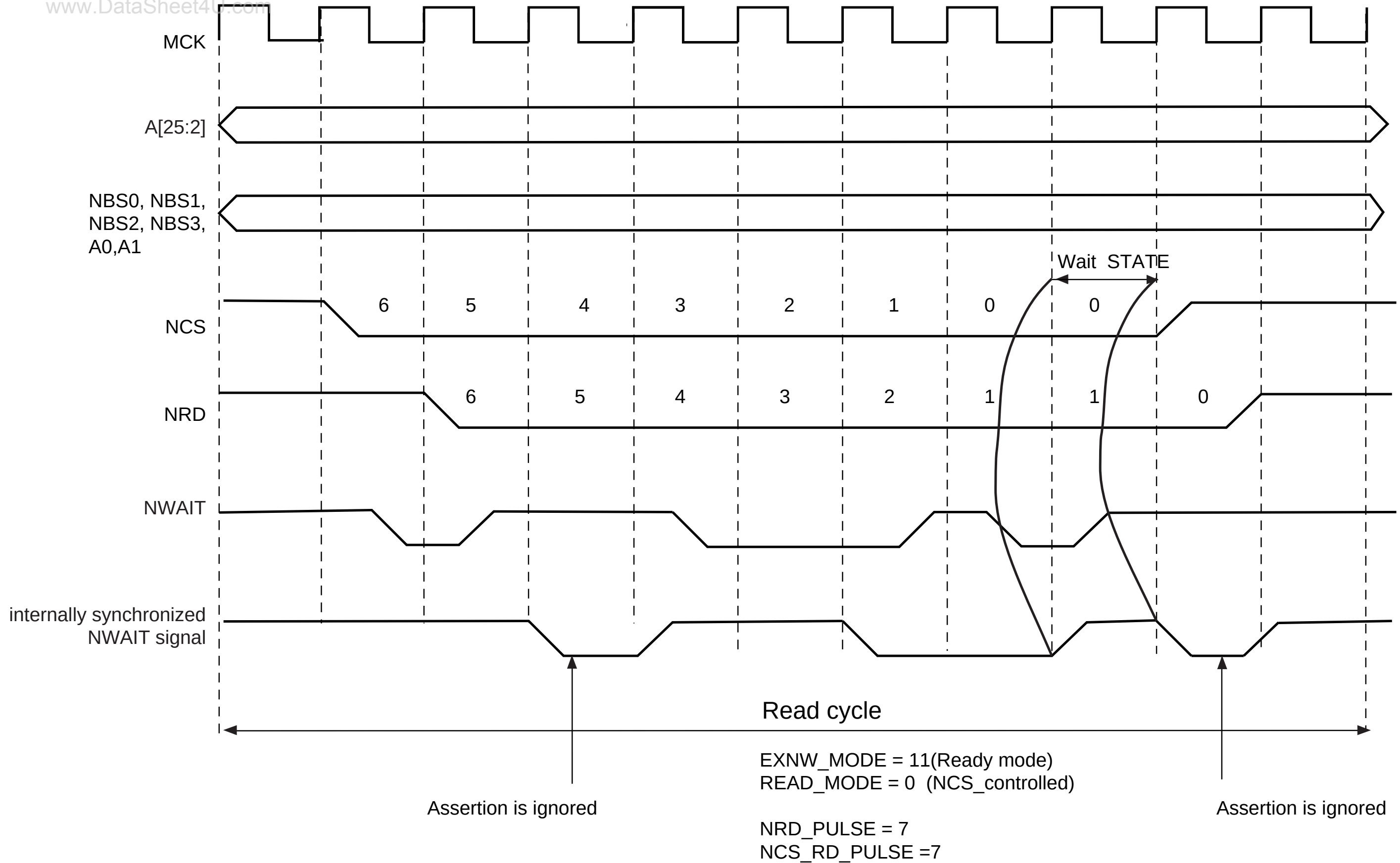


Figure 16-29. NWAIT Assertion in Read Access: Ready Mode (EXNW_MODE = 11)



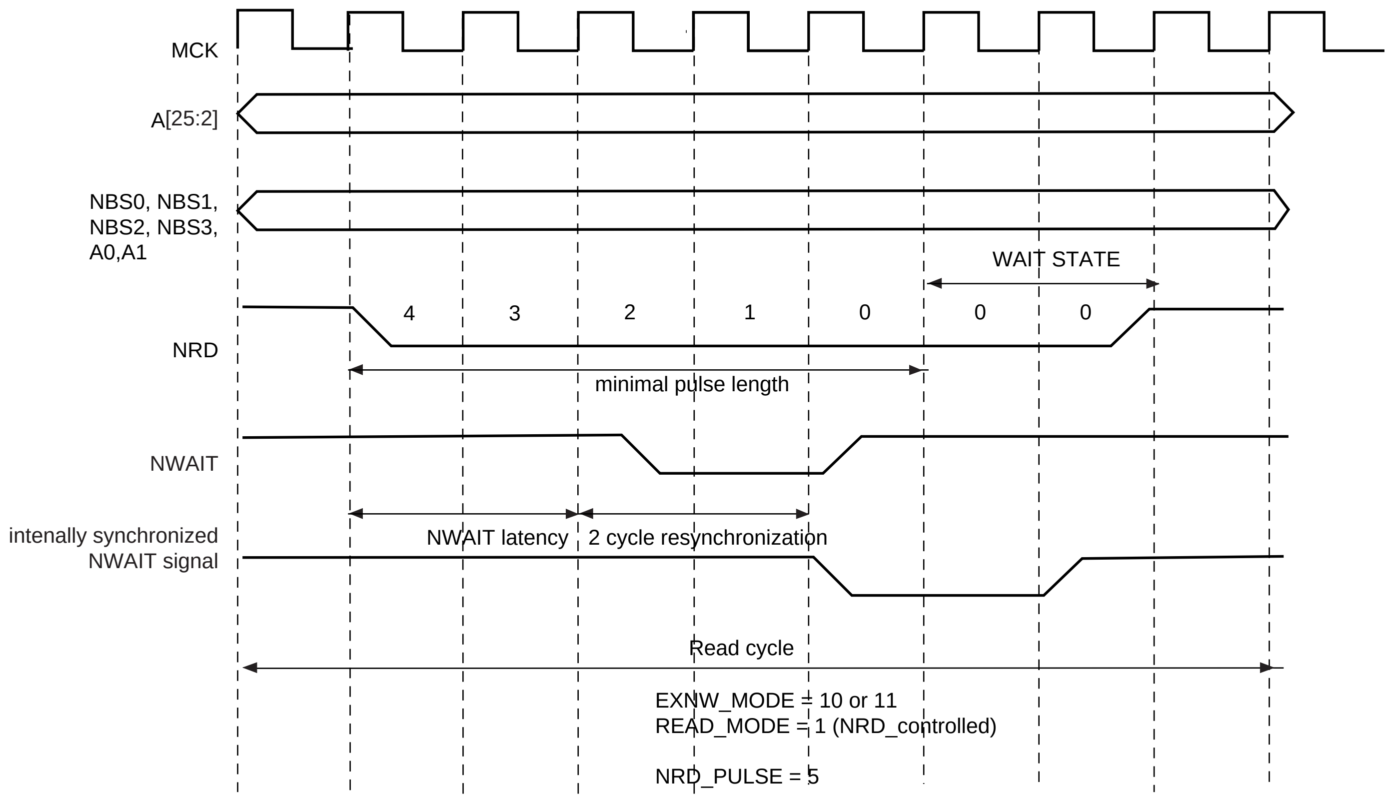
16.11.4 NWAIT Latency and Read/write Timings

There may be a latency between the assertion of the read/write controlling signal and the assertion of the NWAIT signal by the device. The programmed pulse length of the read/write controlling signal must be at least equal to this latency plus the 2 cycles of resynchronization + 1 cycle. Otherwise, the SMC may enter the hold state of the access without detecting the NWAIT signal assertion. This is true in frozen mode as well as in ready mode. This is illustrated on Figure 16-30.

When EXNW_MODE is enabled (ready or frozen), the user must program a pulse length of the read and write controlling signal of at least:

minimal pulse length = NWAIT latency + 2 resynchronization cycles + 1 cycle

Figure 16-30. NWAIT Latency



16.12 Slow Clock Mode

The SMC is able to automatically apply a set of “slow clock mode” read/write waveforms when an internal signal driven by the Power Management Controller is asserted because MCK has been turned to a very slow clock rate (typically 32kHz clock rate). In this mode, the user-programmed waveforms are ignored and the slow clock mode waveforms are applied. This mode is provided so as to avoid reprogramming the User Interface with appropriate waveforms at very slow clock rate. When activated, the slow mode is active on all chip selects.

16.12.1 Slow Clock Mode Waveforms

Figure 16-31 illustrates the read and write operations in slow clock mode. They are valid on all chip selects. Table 16-6 indicates the value of read and write parameters in slow clock mode.

Figure 16-31. Read/write Cycles in Slow Clock Mode

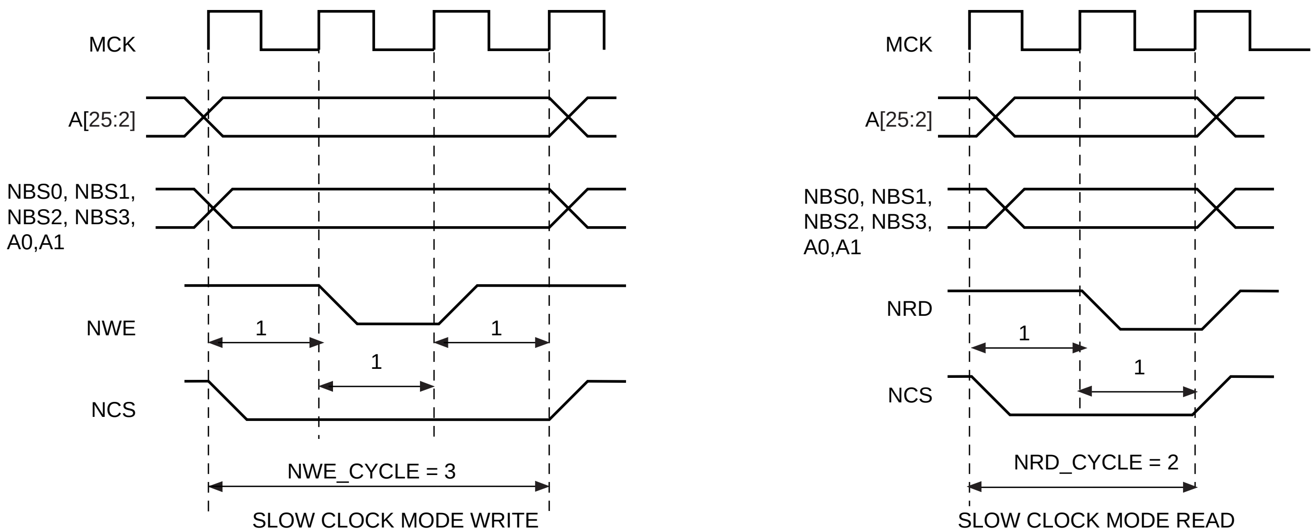


Table 16-6. Read and Write Timing Parameters in Slow Clock Mode

Read Parameters	Duration (cycles)	Write Parameters	Duration (cycles)
NRD_SETUP	1	NWE_SETUP	1
NRD_PULSE	1	NWE_PULSE	1
NCS_RD_SETUP	0	NCS_WR_SETUP	0
NCS_RD_PULSE	2	NCS_WR_PULSE	3
NRD_CYCLE	2	NWE_CYCLE	3

16.12.2 Switching from (to) Slow Clock Mode to (from) Normal Mode

When switching from slow clock mode to the normal mode, the current slow clock mode transfer is completed at high clock rate, with the set of slow clock mode parameters.See Figure 16-32 on page 184. The external device may not be fast enough to support such timings.

Figure 16-33 illustrates the recommended procedure to properly switch from one mode to the other.

Figure 16-32. Clock Rate Transition Occurs while the SMC is Performing a Write Operation

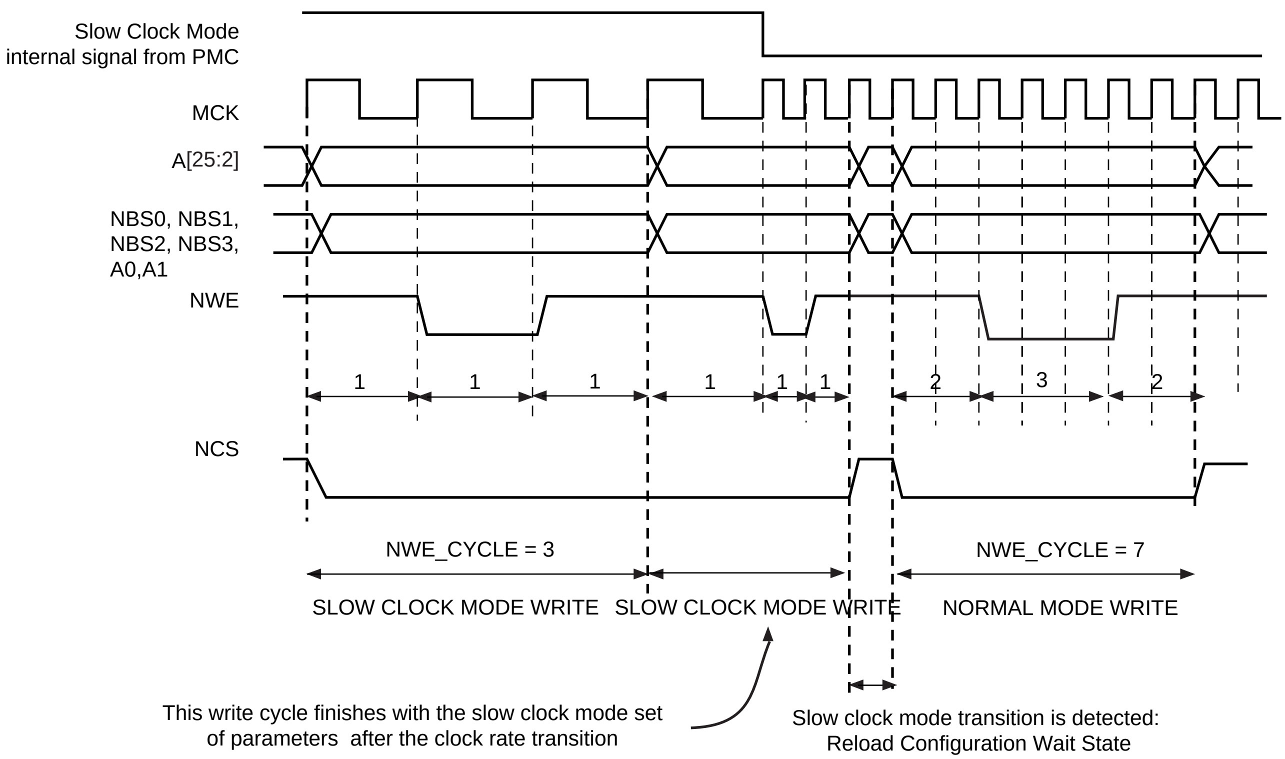
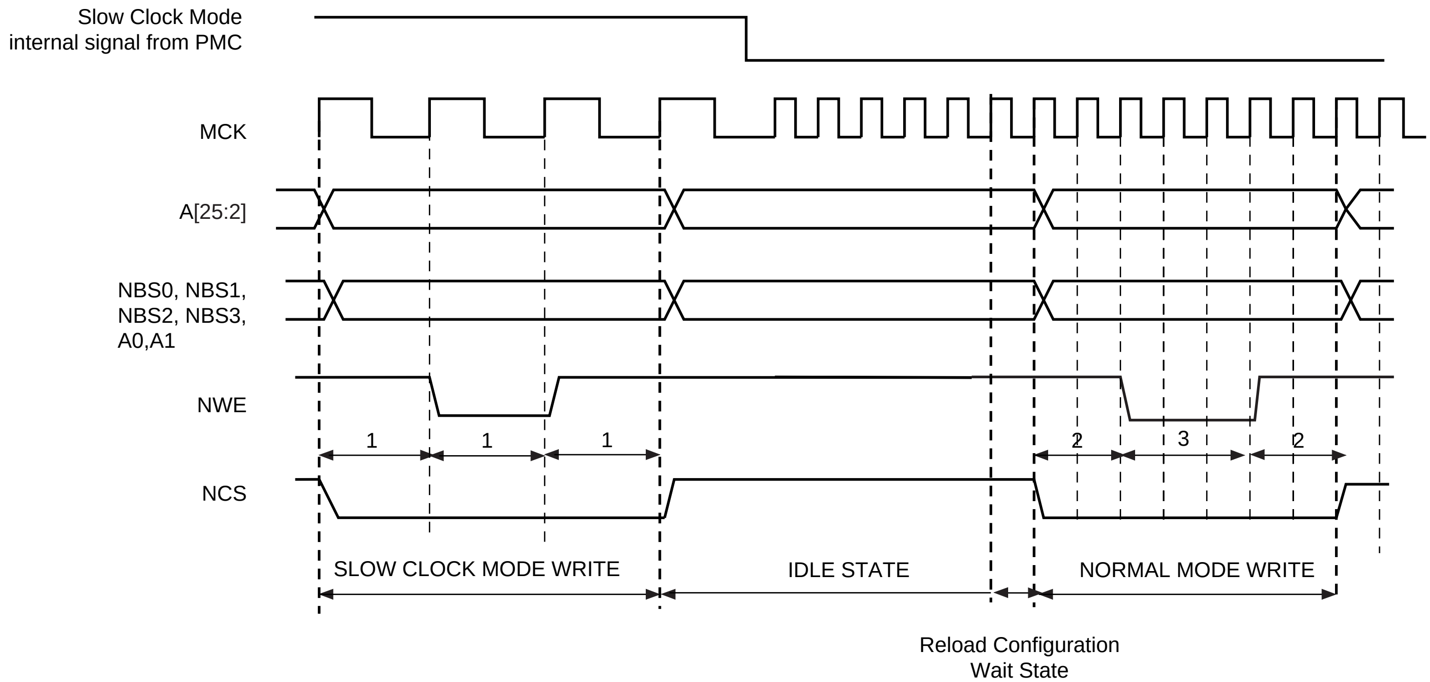


Figure 16-33. Recommended Procedure to Switch from Slow Clock Mode to Normal Mode or from Normal Mode to Slow Clock Mode



16.13 Asynchronous Page Mode

The SMC supports asynchronous burst reads in page mode, providing that the page mode is enabled in the SMC_MODE register (PMEN field). The page size must be configured in the SMC_MODE register (PS field) to 4, 8, 16 or 32 bytes.

The page defines a set of consecutive bytes into memory. A 4-byte page (resp. 8-, 16-, 32-byte page) is always aligned to 4-byte boundaries (resp. 8-, 16-, 32-byte boundaries) of memory. The MSB of data address defines the address of the page in memory, the LSB of address define the address of the data in the page as detailed in Table 16-7.

With page mode memory devices, the first access to one page (t_{pa}) takes longer than the subsequent accesses to the page (t_{sa}) as shown in Figure 16-34. When in page mode, the SMC enables the user to define different read timings for the first access within one page, and next accesses within the page.

Table 16-7. Page Address and Data Address within a Page

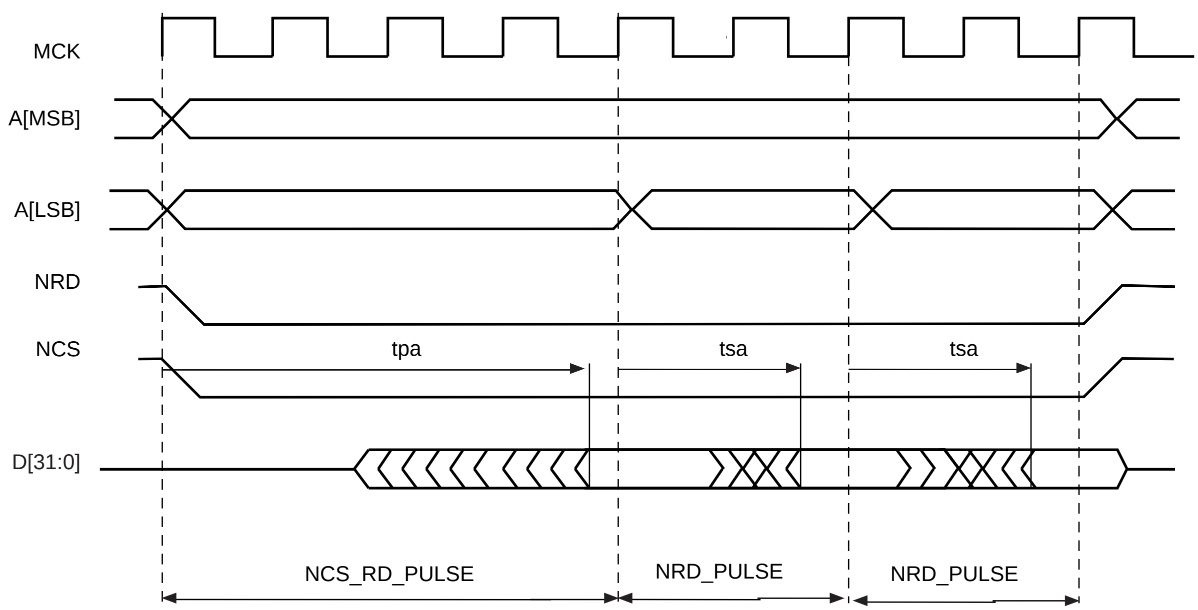
Page Size	**Page Address**[1]	Data Address in the Page[2]
4 bytes	A[25:2]	A[1:0]
8 bytes	A[25:3]	A[2:0]
16 bytes	A[25:4]	A[3:0]
32 bytes	A[25:5]	A[4:0]

Notes: 1. A denotes the address bus of the memory device
2. For 16-bit devices, the bit 0 of address is ignored. For 32-bit devices, bits [1:0] are ignored.

16.13.1 Protocol and Timings in Page Mode

Figure 16-34 shows the NRD and NCS timings in page mode access.

Figure 16-34. Page Mode Read Protocol (Address MSB and LSB are defined in Table 16-7)



The NRD and NCS signals are held low during all read transfers, whatever the programmed values of the setup and hold timings in the User Interface may be. Moreover, the NRD and NCS timings are identical. The pulse length of the first access to the page is defined with the

NCS_RD_PULSE field of the SMC_PULSE register. The pulse length of subsequent accesses within the page are defined using the NRD_PULSE parameter.

In page mode, the programming of the read timings is described in Table 16-8:

Table 16-8. Programming of Read Timings in Page Mode

Parameter	Value	Definition
READ_MODE	'x'	No impact
NCS_RD_SETUP	'x'	No impact
NCS_RD_PULSE	t_{pa}	Access time of first access to the page
NRD_SETUP	'x'	No impact
NRD_PULSE	t_{sa}	Access time of subsequent accesses in the page
NRD_CYCLE	'x'	No impact

The SMC does not check the coherency of timings. It will always apply the NCS_RD_PULSE timings as page access timing (t_{pa}) and the NRD_PULSE for accesses to the page (t_{sa}), even if the programmed value for t_{pa} is shorter than the programmed value for t_{sa}.

16.13.2 Byte Access Type in Page Mode

The Byte Access Type configuration remains active in page mode. For 16-bit or 32-bit page mode devices that require byte selection signals, configure the BAT field of the SMC_REGISTER to 0 (byte select access type).

16.13.3 Page Mode Restriction

The page mode is not compatible with the use of the NWAIT signal. Using the page mode and the NWAIT signal may lead to unpredictable behavior.

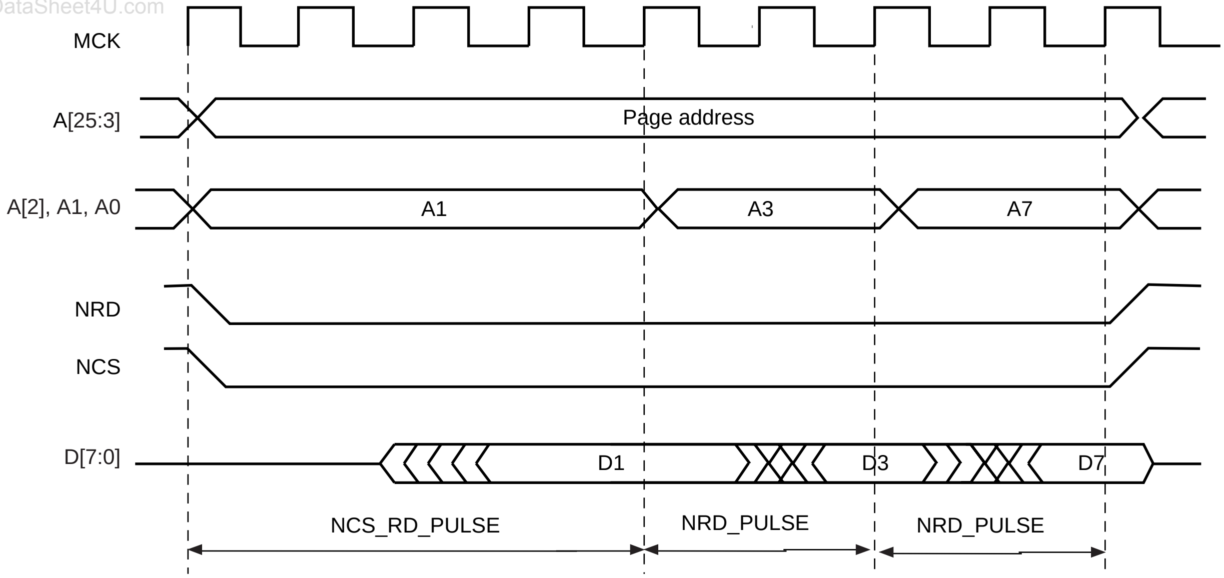
16.13.4 Sequential and Non-sequential Accesses

If the chip select and the MSB of addresses as defined in Table 16-7 are identical, then the current access lies in the same page as the previous one, and no page break occurs.

Using this information, all data within the same page, sequential or not sequential, are accessed with a minimum access time (t_{sa}). Figure 16-35 illustrates access to an 8-bit memory device in page mode, with 8-byte pages. Access to D1 causes a page access with a long access time (t_{pa}). Accesses to D3 and D7, though they are not sequential accesses, only require a short access time (t_{sa}).

If the MSB of addresses are different, the SMC performs the access of a new page. In the same way, if the chip select is different from the previous access, a page break occurs. If two sequential accesses are made to the page mode memory, but separated by an other internal or external peripheral access, a page break occurs on the second access because the chip select of the device was deasserted between both accesses.

Figure 16-35. Access to Non-sequential Data within the Same Page



16.14 Static Memory Controller (SMC) User Interface

The SMC is programmed using the registers listed in Table 16-9. For each chip select, a set of 4 registers is used to program the parameters of the external device connected on it. In Table 16-9, "CS_number" denotes the chip select number. 16 bytes (0x10) are required per chip select.

The user must complete writing the configuration by writing any one of the SMC_MODE registers.

Table 16-9. SMC Register Mapping

Offset	Register	Name	Access	Reset State
0x10 x CS_number + 0x00	SMC Setup Register	SMC_SETUP	Read/Write	0x00000000
0x10 x CS_number + 0x04	SMC Pulse Register	SMC_PULSE	Read/Write	0x01010101
0x10 x CS_number + 0x08	SMC Cycle Register	SMC_CYCLE	Read/Write	0x00010001
0x10 x CS_number + 0x0C	SMC Mode Register	SMC_MODE	Read/Write	0x10002000

16.14.1 SMC Setup Register

Register Name: SMC_SETUP[0 ..7]

Access Type: Read/Write

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

- **NWE_SETUP: NWE Setup Length**

The NWE signal setup length is defined as:

NWE setup length = (128* NWE_SETUP[5] + NWE_SETUP[4:0]) clock cycles

- **NCS_WR_SETUP: NCS Setup Length in WRITE Access**

In write access, the NCS signal setup length is defined as:

NCS setup length = (128* NCS_WR_SETUP[5] + NCS_WR_SETUP[4:0]) clock cycles

- **NRD_SETUP: NRD Setup Length**

The NRD signal setup length is defined in clock cycles as:

NRD setup length = (128* NRD_SETUP[5] + NRD_SETUP[4:0]) clock cycles

- **NCS_RD_SETUP: NCS Setup Length in READ Access**

In read access, the NCS signal setup length is defined as:

NCS setup length = (128* NCS_RD_SETUP[5] + NCS_RD_SETUP[4:0]) clock cycles

16.14.2 SMC Pulse Register

Register Name: SMC_PULSE[0..7]

Access Type: Read/Write

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

- **NWE_PULSE: NWE Pulse Length**

The NWE signal pulse length is defined as:

NWE pulse length = (256* NWE_PULSE[6] + NWE_PULSE[5:0]) clock cycles

The NWE pulse length must be at least 1 clock cycle.

- **NCS_WR_PULSE: NCS Pulse Length in WRITE Access**

In write access, the NCS signal pulse length is defined as:

NCS pulse length = (256* NCS_WR_PULSE[6] + NCS_WR_PULSE[5:0]) clock cycles

The NCS pulse length must be at least 1 clock cycle.

- **NRD_PULSE: NRD Pulse Length**

In standard read access, the NRD signal pulse length is defined in clock cycles as:

NRD pulse length = (256* NRD_PULSE[6] + NRD_PULSE[5:0]) clock cycles

The NRD pulse length must be at least 1 clock cycle.

In page mode read access, the NRD_PULSE parameter defines the duration of the subsequent accesses in the page.

- **NCS_RD_PULSE: NCS Pulse Length in READ Access**

In standard read access, the NCS signal pulse length is defined as:

NCS pulse length = (256* NCS_RD_PULSE[6] + NCS_RD_PULSE[5:0]) clock cycles

The NCS pulse length must be at least 1 clock cycle.

In page mode read access, the NCS_RD_PULSE parameter defines the duration of the first access to one page.

16.14.3 SMC Cycle Register

Register Name: SMC_CYCLE[0..7]

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
NRD_CYCLE							

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

7	6	5	4	3	2	1	0
NWE_CYCLE							

- **NWE_CYCLE: Total Write Cycle Length**

The total write cycle length is the total duration in clock cycles of the write cycle. It is equal to the sum of the setup, pulse and hold steps of the NWE and NCS signals. It is defined as:

Write cycle length = (NWE_CYCLE[8:7]*256 + NWE_CYCLE[6:0]) clock cycles

- **NRD_CYCLE: Total Read Cycle Length**

The total read cycle length is the total duration in clock cycles of the read cycle. It is equal to the sum of the setup, pulse and hold steps of the NRD and NCS signals. It is defined as:

Read cycle length = (NRD_CYCLE[8:7]*256 + NRD_CYCLE[6:0]) clock cycles

16.14.4 SMC MODE Register

Register Name: SMC_MODE[0..7]

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	DBW		–	–	–	BAT

7	6	5	4	3	2	1	0
–	–	EXNW_MODE		–	–	WRITE_MODE	READ_MODE

- **READ_MODE:**

1: The read operation is controlled by the NRD signal.

– If TDF cycles are programmed, the external bus is marked busy after the rising edge of NRD.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states are inserted after the setup of NRD.

0: The read operation is controlled by the NCS signal.

– If TDF cycles are programmed, the external bus is marked busy after the rising edge of NCS.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states are inserted after the setup of NCS.

- **WRITE_MODE**

1: The write operation is controlled by the NWE signal.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states will be inserted after the setup of NWE.

0: The write operation is controlled by the NCS signal.

– If TDF optimization is enabled (TDF_MODE =1), TDF wait states will be inserted after the setup of NCS.

- **EXNW_MODE: NWAIT Mode**

The NWAIT signal is used to extend the current read or write signal. It is only taken into account during the pulse phase of the read and write controlling signal. When the use of NWAIT is enabled, at least one cycle hold duration must be programmed for the read and write controlling signal.

EXNW_MODE		NWAIT Mode
0	0	Disabled
0	1	Reserved
1	0	Frozen Mode
1	1	Ready Mode

- Disabled Mode: The NWAIT input signal is ignored on the corresponding Chip Select.
- Frozen Mode: If asserted, the NWAIT signal freezes the current read or write cycle. After deassertion, the read/write cycle is resumed from the point where it was stopped.

- Ready Mode: The NWAIT signal indicates the availability of the external device at the end of the pulse of the controlling read or write signal, to complete the access. If high, the access normally completes. If low, the access is extended until NWAIT returns high.

- **BAT: Byte Access Type**

This field is used only if DBW defines a 16- or 32-bit data bus.

- 1: Byte write access type:
  – Write operation is controlled using NCS, NWR0, NWR1, NWR2, NWR3.
  – Read operation is controlled using NCS and NRD.
- 0: Byte select access type:
  – Write operation is controlled using NCS, NWE, NBS0, NBS1, NBS2 and NBS3
  – Read operation is controlled using NCS, NRD, NBS0, NBS1, NBS2 and NBS3

- **DBW: Data Bus Width**

DBW		Data Bus Width
0	0	8-bit bus
0	1	16-bit bus
1	0	32-bit bus
1	1	Reserved

- **TDF_CYCLES: Data Float Time**

This field gives the integer number of clock cycles required by the external device to release the data after the rising edge of the read controlling signal. The SMC always provide one full cycle of bus turnaround after the TDF_CYCLES period. The external bus cannot be used by another chip select during TDF_CYCLES + 1 cycles. From 0 up to 15 TDF_CYCLES can be set.

- **TDF_MODE: TDF Optimization**

1: TDF optimization is enabled.

  – The number of TDF wait states is optimized using the setup period of the next read/write access.

0: TDF optimization is disabled.

  – The number of TDF wait states is inserted before the next access begins.

- **PMEN: Page Mode Enabled**

1: Asynchronous burst read in page mode is applied on the corresponding chip select.

0: Standard read is applied.

- **PS: Page Size**

If page mode is enabled, this field indicates the size of the page in bytes.

PS		Page Size
0	0	4-byte page
0	1	8-byte page
1	0	16-byte page
1	1	32-byte page

17. SDRAM Controller (SDRAMC)

17.1 Description

The SDRAM Controller (SDRAMC) extends the memory capabilities of a chip by providing the interface to an external 16-bit or 32-bit SDRAM device. The page size supports ranges from 2048 to 8192 and the number of columns from 256 to 2048. It supports byte (8-bit), half-word (16-bit) and word (32-bit) accesses.

The SDRAM Controller supports a read or write burst length of one location. It keeps track of the active row in each bank, thus maximizing SDRAM performance, e.g., the application may be placed in one bank and data in the other banks. So as to optimize performance, it is advisable to avoid accessing different rows in the same bank.

The SDRAM controller supports a CAS latency of 1, 2 or 3 and optimizes the read access depending on the frequency.

The different modes available - self-refresh, power-down and deep power-down modes - minimize power consumption on the SDRAM device.

17.2 I/O Lines Description

Table 17-1. I/O Line Description

Name	Description	Type	Active Level
SDCK	SDRAM Clock	Output	
SDCKE	SDRAM Clock Enable	Output	High
SDCS	SDRAM Controller Chip Select	Output	Low
BA[1:0]	Bank Select Signals	Output	
RAS	Row Signal	Output	Low
CAS	Column Signal	Output	Low
SDWE	SDRAM Write Enable	Output	Low
NBS[3:0]	Data Mask Enable Signals	Output	Low
SDRAMC_A[12:0]	Address Bus	Output	
D[31:0]	Data Bus	I/O	

17.3 Application Example

17.3.1 Software Interface

The SDRAM address space is organized into banks, rows, and columns. The SDRAM controller allows mapping different memory types according to the values set in the SDRAMC configuration register.

The SDRAM Controller’s function is to make the SDRAM device access protocol transparent to the user. Table 17-2 to Table 17-7 illustrate the SDRAM device memory mapping seen by the user in correlation with the device structure. Various configurations are illustrated.

17.3.1.1 32-bit Memory Data Bus Width

Table 17-2. SDRAM Configuration Mapping: 2K Rows, 256/512/1024/2048 Columns

CPU Address Line																											
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**
					Bk[1:0]		Row[10:0]											Column[7:0]								M[1:0]	
				Bk[1:0]		Row[10:0]											Column[8:0]									M[1:0]	
			Bk[1:0]		Row[10:0]											Column[9:0]										M[1:0]	
		Bk[1:0]		Row[10:0]											Column[10:0]											M[1:0]	

Table 17-3. SDRAM Configuration Mapping: 4K Rows, 256/512/1024/2048 Columns

CPU Address Line																											
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**
				Bk[1:0]		Row[11:0]												Column[7:0]								M[1:0]	
			Bk[1:0]		Row[11:0]												Column[8:0]									M[1:0]	
		Bk[1:0]		Row[11:0]												Column[9:0]										M[1:0]	
	Bk[1:0]		Row[11:0]												Column[10:0]											M[1:0]	

Table 17-4. SDRAM Configuration Mapping: 8K Rows, 256/512/1024/2048 Columns

CPU Address Line																											
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**
			Bk[1:0]		Row[12:0]													Column[7:0]								M[1:0]	
		Bk[1:0]		Row[12:0]													Column[8:0]									M[1:0]	
	Bk[1:0]		Row[12:0]													Column[9:0]										M[1:0]	
Bk[1:0]		Row[12:0]													Column[10:0]											M[1:0]	

Notes: 1. M[1:0] is the byte address inside a 32-bit word.
2. Bk[1] = BA1, Bk[0] = BA0.

17.3.1.2 16-bit Memory Data Bus Width

Table 17-5. SDRAM Configuration Mapping: 2K Rows, 256/512/1024/2048 Columns

CPU Address Line																											
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**
						Bk[1:0]		Row[10:0]											Column[7:0]								M0
					Bk[1:0]		Row[10:0]											Column[8:0]									M0
				Bk[1:0]		Row[10:0]											Column[9:0]										M0
			Bk[1:0]		Row[10:0]											Column[10:0]											M0

Table 17-6. SDRAM Configuration Mapping: 4K Rows, 256/512/1024/2048 Columns

CPU Address Line																											
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**
					Bk[1:0]		Row[11:0]												Column[7:0]								M0
				Bk[1:0]		Row[11:0]												Column[8:0]									M0
			Bk[1:0]		Row[11:0]												Column[9:0]										M0
		Bk[1:0]		Row[11:0]												Column[10:0]											M0

Table 17-7. SDRAM Configuration Mapping: 8K Rows, 256/512/1024/2048 Columns

CPU Address Line																											
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**
				Bk[1:0]		Row[12:0]													Column[7:0]								M0
			Bk[1:0]		Row[12:0]													Column[8:0]									M0
		Bk[1:0]		Row[12:0]													Column[9:0]										M0
	Bk[1:0]		Row[12:0]													Column[10:0]											M0

Notes:
1. M0 is the byte address inside a 16-bit half-word.
2. Bk[1] = BA1, Bk[0] = BA0.

17.4 Product Dependencies

17.4.1 SDRAM Device Initialization

The initialization sequence is generated by software. The SDRAM devices are initialized by the following sequence:

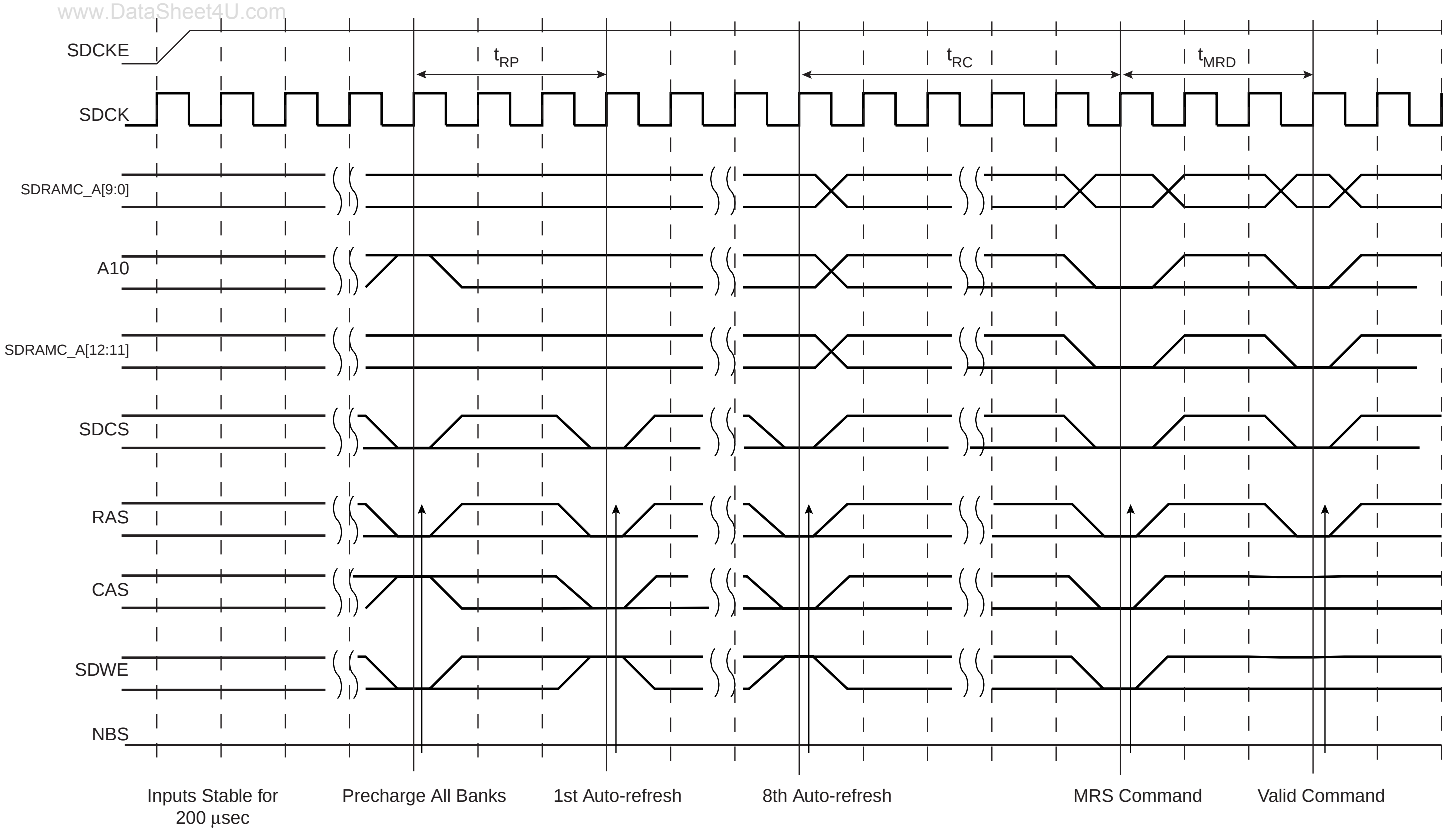
1. SDRAM features must be set in the configuration register: asynchronous timings (TRC, TRAS, etc.), number of columns, rows, CAS latency, and the data bus width.
2. For mobile SDRAM, temperature-compensated self refresh (TCSR), drive strength (DS) and partial array self refresh (PASR) must be set in the Low Power Register.
3. The SDRAM memory type must be set in the Memory Device Register.
4. A minimum pause of 200 µs is provided to precede any signal toggle.
5. [(1)]A NOP command is issued to the SDRAM devices. The application must set Mode to 1 in the Mode Register and perform a write access to any SDRAM address.

6. An All Banks Precharge command is issued to the SDRAM devices. The application must set Mode to 2 in the Mode Register and perform a write access to any SDRAM address.
7. Eight auto-refresh (CBR) cycles are provided. The application must set the Mode to 4 in the Mode Register and perform a write access to any SDRAM location eight times.
8. A Mode Register set (MRS) cycle is issued to program the parameters of the SDRAM devices, in particular CAS latency and burst length. The application must set Mode to 3 in the Mode Register and perform a write access to the SDRAM. The write address must be chosen so that BA[1:0] are set to 0. For example, with a 16-bit 128 MB SDRAM (12 rows, 9 columns, 4 banks) bank address, the SDRAM write access should be done at the address 0x20000000.
9. For mobile SDRAM initialization, an Extended Mode Register set (EMRS) cycle is issued to program the SDRAM parameters (TCSR, PASR, DS). The application must set Mode to 5 in the Mode Register and perform a write access to the SDRAM. The write address must be chosen so that BA[1] or BA[0] are set to 1. For example, with a 16-bit 128 MB SDRAM, (12 rows, 9 columns, 4 banks) bank address the SDRAM write access should be done at the address 0x20800000 or 0x20400000.
10. The application must go into Normal Mode, setting Mode to 0 in the Mode Register and performing a write access at any location in the SDRAM.
11. Write the refresh rate into the count field in the SDRAMC Refresh Timer register. (Refresh rate = delay between refresh cycles). The SDRAM device requires a refresh every 15.625 µs or 7.81 µs. With a 100 MHz frequency, the Refresh Timer Counter Register must be set with the value 1562(15.652 µs x 100 MHz) or 781(7.81 µs x 100 MHz).

After initialization, the SDRAM devices are fully functional.

Note: 1. It is strongly recommended to respect the instructions stated in Step 5 of the initialization process in order to be certain that the subsequent commands issued by the SDRAMC will be taken into account.

Figure 17-1. SDRAM Device Initialization Sequence



17.4.2 I/O Lines

The pins used for interfacing the SDRAM Controller may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the SDRAM Controller pins to their peripheral function. If I/O lines of the SDRAM Controller are not used by the application, they can be used for other purposes by the PIO Controller.

17.4.3 Interrupt

The SDRAM Controller interrupt (Refresh Error notification) is connected to the Memory Controller. This interrupt may be ORed with other System Peripheral interrupt lines and is finally provided as the System Interrupt Source (Source 1) to the AIC (Advanced Interrupt Controller).

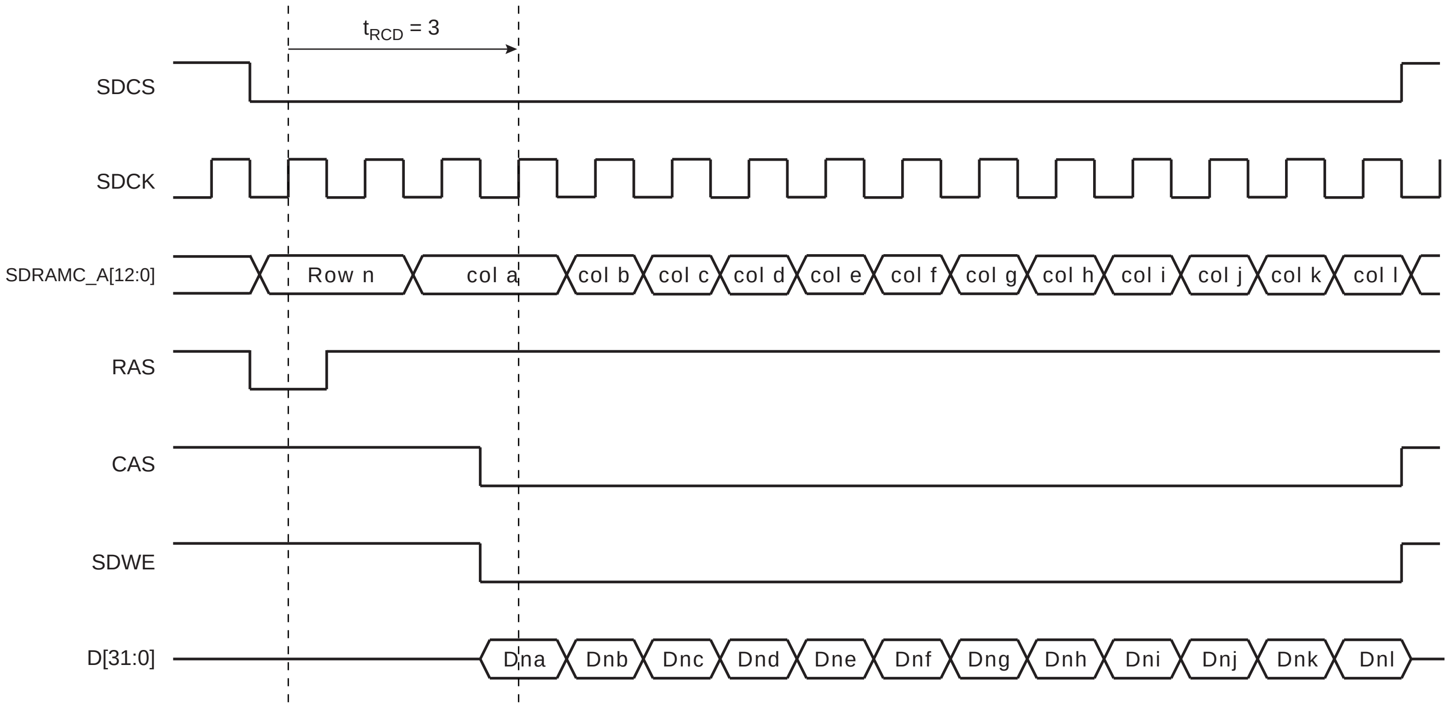
Using the SDRAM Controller interrupt requires the AIC to be programmed first.

17.5 Functional Description

17.5.1 SDRAM Controller Write Cycle

The SDRAM Controller allows burst access or single access. In both cases, the SDRAM controller keeps track of the active row in each bank, thus maximizing performance. To initiate a burst access, the SDRAM Controller uses the transfer type signal provided by the master requesting the access. If the next access is a sequential write access, writing to the SDRAM device is carried out. If the next access is a write-sequential access, but the current access is to a boundary page, or if the next access is in another row, then the SDRAM Controller generates a precharge command, activates the new row and initiates a write command. To comply with SDRAM timing parameters, additional clock cycles are inserted between precharge/active (t_{RP}) commands and active/write (t_{RCD}) commands. For definition of these timing parameters, refer to the "SDRAMC Configuration Register" on page 210. This is described in Figure 17-2 below.

Figure 17-2. Write Burst, 32-bit SDRAM Access



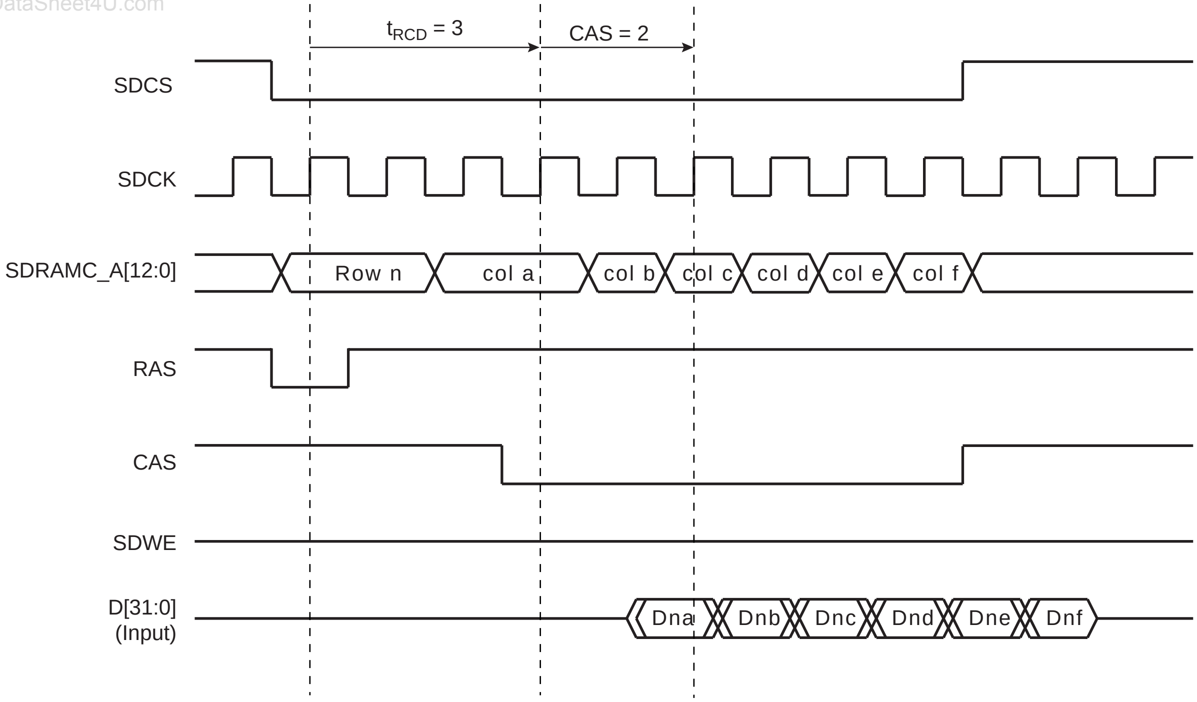
17.5.2 SDRAM Controller Read Cycle

The SDRAM Controller allows burst access, incremental burst of unspecified length or single access. In all cases, the SDRAM Controller keeps track of the active row in each bank, thus maximizing performance of the SDRAM. If row and bank addresses do not match the previous row/bank address, then the SDRAM controller automatically generates a precharge command, activates the new row and starts the read command. To comply with the SDRAM timing parameters, additional clock cycles on SDCK are inserted between precharge and active commands (t_{RP}) and between active and read command (t_{RCD}). These two parameters are set in the configuration register of the SDRAM Controller. After a read command, additional wait states are generated to comply with the CAS latency (1, 2 or 3 clock delays specified in the configuration register).

For a single access or an incremented burst of unspecified length, the SDRAM Controller anticipates the next access. While the last value of the column is returned by the SDRAM Controller on the bus, the SDRAM Controller anticipates the read to the next column and thus anticipates the CAS latency. This reduces the effect of the CAS latency on the internal bus.

For burst access of specified length (4, 8, 16 words), access is not anticipated. This case leads to the best performance. If the burst is broken (border, busy mode, etc.), the next access is handled as an incrementing burst of unspecified length.

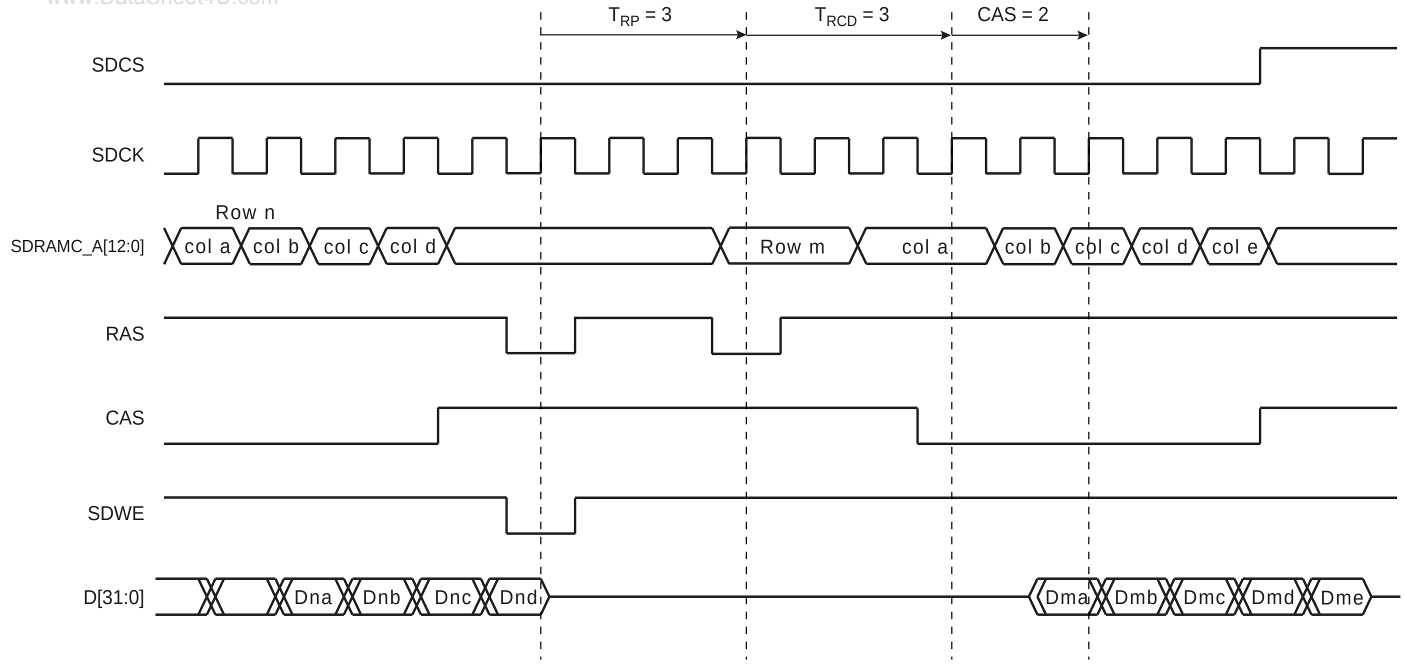
Figure 17-3. Read Burst, 32-bit SDRAM Access



17.5.3 Border Management

When the memory row boundary has been reached, an automatic page break is inserted. In this case, the SDRAM controller generates a precharge command, activates the new row and initiates a read or write command. To comply with SDRAM timing parameters, an additional clock cycle is inserted between the precharge/active (t_{RP}) command and the active/read (t_{RCD}) command. This is described in Figure 17-4 below.

Figure 17-4. Read Burst with Boundary Row Access



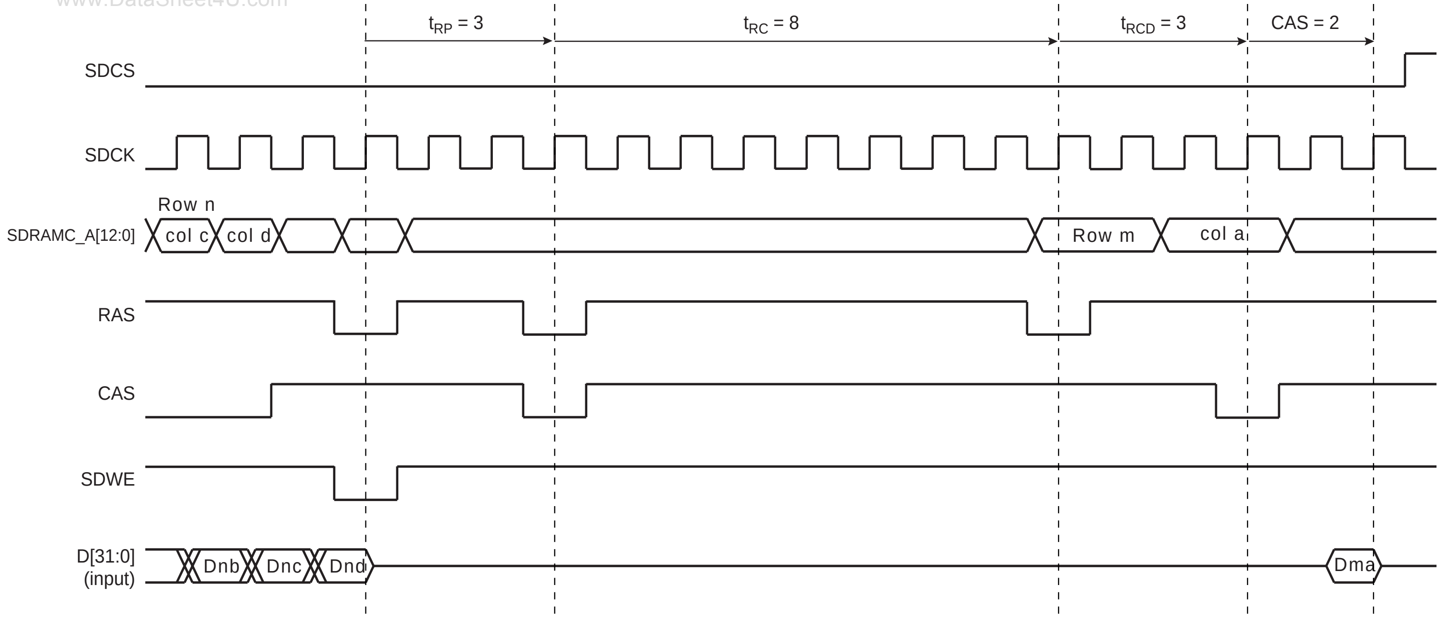
17.5.4 SDRAM Controller Refresh Cycles

An auto-refresh command is used to refresh the SDRAM device. Refresh addresses are generated internally by the SDRAM device and incremented after each auto-refresh automatically. The SDRAM Controller generates these auto-refresh commands periodically. An internal timer is loaded with the value in the register SDRAMC_TR that indicates the number of clock cycles between refresh cycles.

A refresh error interrupt is generated when the previous auto-refresh command did not perform. It is acknowledged by reading the Interrupt Status Register (SDRAMC_ISR).

When the SDRAM Controller initiates a refresh of the SDRAM device, internal memory accesses are not delayed. However, if the CPU tries to access the SDRAM, the slave indicates that the device is busy and the master is held by a wait signal. See Figure 17-5.

Figure 17-5. Refresh Cycle Followed by a Read Access



17.5.5 Power Management

Three low-power modes are available:

- Self-refresh Mode: The SDRAM executes its own Auto-refresh cycle without control of the SDRAM Controller. Current drained by the SDRAM is very low.
- Power-down Mode: Auto-refresh cycles are controlled by the SDRAM Controller. Between auto-refresh cycles, the SDRAM is in power-down. Current drained in Power-down mode is higher than in Self-refresh Mode.
- Deep Power-down Mode: (Only available with Mobile SDRAM) The SDRAM contents are lost, but the SDRAM does not drain any current.

The SDRAM Controller activates one low-power mode as soon as the SDRAM device is not selected. It is possible to delay the entry in self-refresh and power-down mode after the last access by programming a timeout value in the Low Power Register.

17.5.5.1 Self-refresh Mode

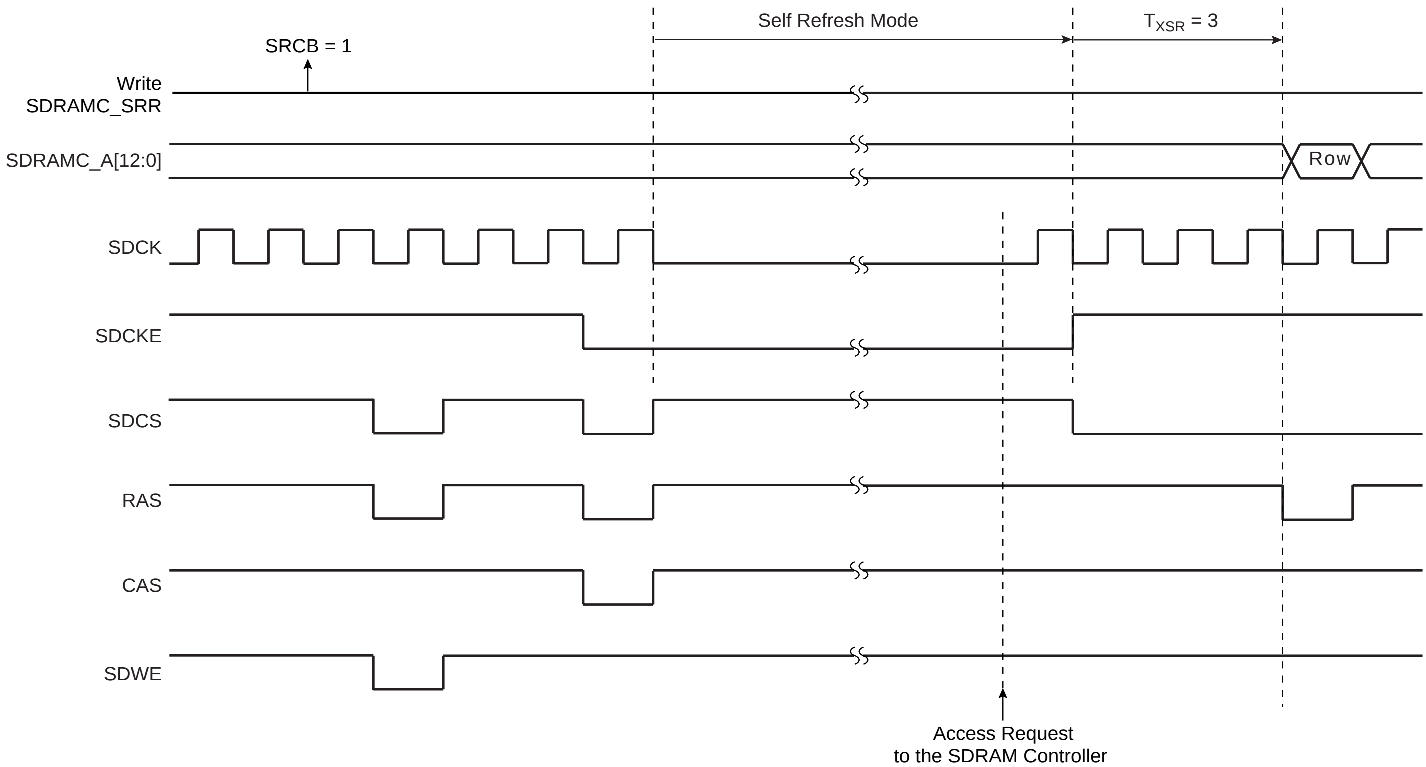
This mode is selected by programming the LPCB field to 1 in the SDRAMC Low Power Register. In self-refresh mode, the SDRAM device retains data without external clocking and provides its own internal clocking, thus performing its own auto-refresh cycles. All the inputs to the SDRAM device become “don’t care” except SDCKE, which remains low. As soon as the SDRAM device is selected, the SDRAM Controller provides a sequence of commands and exits self-refresh mode.

Some low-power SDRAMs (e.g., mobile SDRAM) can refresh only one quarter or a half quarter or all banks of the SDRAM array. This feature reduces the self-refresh current. To configure this feature, Temperature Compensated Self Refresh (TCSR), Partial Array Self Refresh (PASR) and Drive Strength (DS) parameters must be set in the Low Power Register and transmitted to the low-power SDRAM during initialization.

After initialization, as soon as PASR/DS/TCSR fields are modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and PASR/DS/TCSR bits are updated before entry into self-refresh mode.

The SDRAM device must remain in self-refresh mode for a minimum period of t_{RAS} and may remain in self-refresh mode for an indefinite period. This is described in Figure 17-6.

Figure 17-6. Self-refresh Mode Behavior

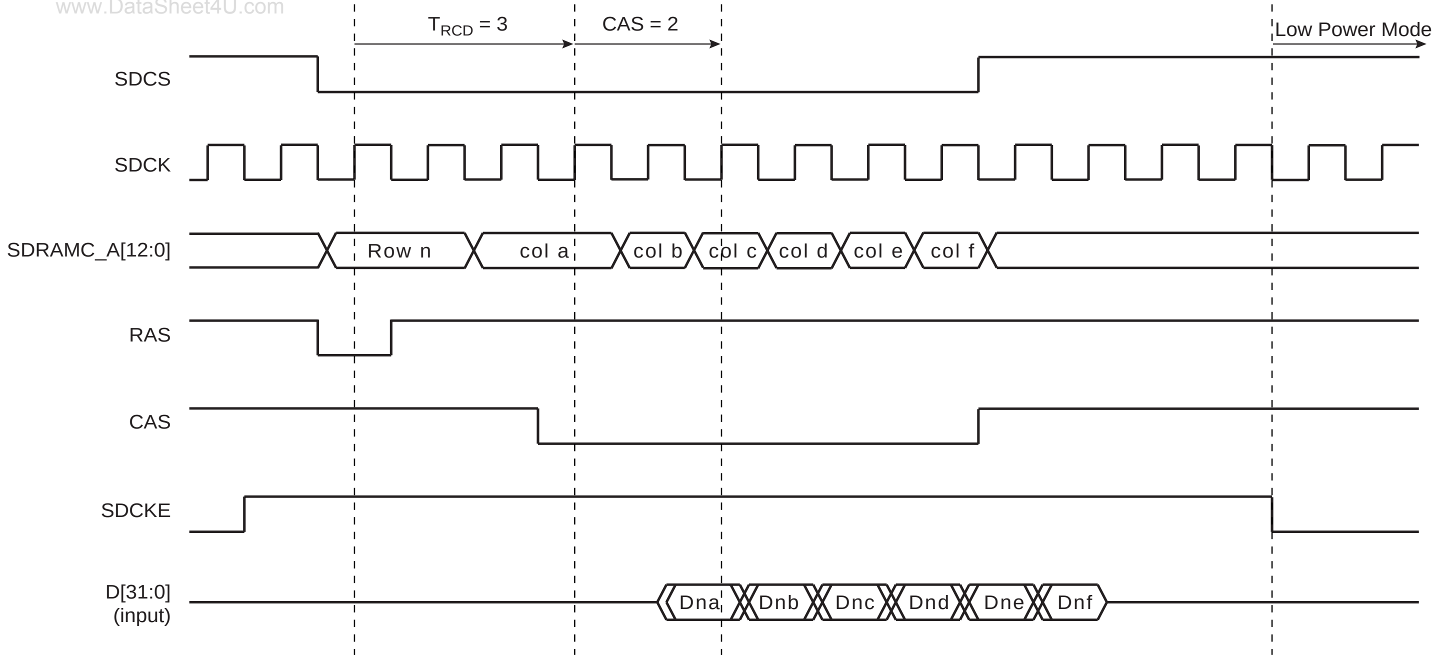


17.5.6 Low-power Mode

This mode is selected by programming the LPCB field to 2 in the SDRAMC Low Power Register. Power consumption is greater than in self-refresh mode. All the input and output buffers of the SDRAM device are deactivated except SDCKE, which remains low. In contrast to self-refresh mode, the SDRAM device cannot remain in low-power mode longer than the refresh period (64 ms for a whole device refresh operation). As no auto-refresh operations are performed by the SDRAM itself, the SDRAM Controller carries out the refresh operation. The exit procedure is faster than in self-refresh mode.

This is described in Figure 17-7.

Figure 17-7. Low-power Mode Behavior



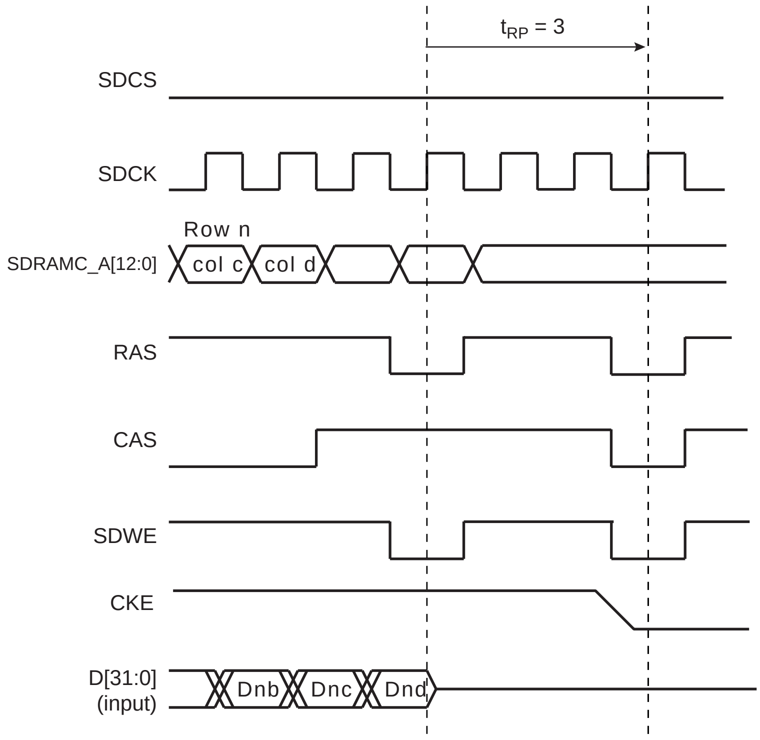
17.5.6.1 Deep Power-down Mode

This mode is selected by programming the LPCB field to 3 in the SDRAMC Low Power Register. When this mode is activated, all internal voltage generators inside the SDRAM are stopped and all data is lost.

When this mode is enabled, the application must not access to the SDRAM until a new initialization sequence is done (See "SDRAM Device Initialization" on page 197).

This is described in Figure 17-8.

Figure 17-8. Deep Power-down Mode Behavior



17.6 SDRAM Controller User Interface

Table 17-8. SDRAM Controller Memory Map

Offset	Register	Name	Access	Reset State
0x00	SDRAMC Mode Register	SDRAMC_MR	Read/Write	0x00000000
0x04	SDRAMC Refresh Timer Register	SDRAMC_TR	Read/Write	0x00000000
0x08	SDRAMC Configuration Register	SDRAMC_CR	Read/Write	0x852372C0
0x0C	SDRAMC High Speed Register	SDRAMC_HSR	Read/Write	0x00
0x10	SDRAMC Low Power Register	SDRAMC_LPR	Read/Write	0x0
0x14	SDRAMC Interrupt Enable Register	SDRAMC_IER	Write-only	–
0x18	SDRAMC Interrupt Disable Register	SDRAMC_IDR	Write-only	–
0x1C	SDRAMC Interrupt Mask Register	SDRAMC_IMR	Read-only	0x0
0x20	SDRAMC Interrupt Status Register	SDRAMC_ISR	Read-only	0x0
0x24	SDRAMC Memory Device Register	SDRAMC_MDR	Read	0x0
0x28 - 0xFC	Reserved	–	–	–

17.6.1 SDRAMC Mode Register

Register Name: SDRAMC_MR

Access Type: Read/Write

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	MODE		

- **MODE: SDRAMC Command Mode**

This field defines the command issued by the SDRAM Controller when the SDRAM device is accessed.

MODE			Description
0	0	0	Normal mode. Any access to the SDRAM is decoded normally.
0	0	1	The SDRAM Controller issues a NOP command when the SDRAM device is accessed regardless of the cycle.
0	1	0	The SDRAM Controller issues an “All Banks Precharge” command when the SDRAM device is accessed regardless of the cycle.
0	1	1	The SDRAM Controller issues a “Load Mode Register” command when the SDRAM device is accessed regardless of the cycle. The address offset with respect to the SDRAM device base address is used to program the Mode Register. For instance, when this mode is activated, an access to the “SDRAM_Base + offset” address generates a “Load Mode Register” command with the value “offset” written to the SDRAM device Mode Register.
1	0	0	The SDRAM Controller issues an “Auto-Refresh” Command when the SDRAM device is accessed regardless of the cycle. Previously, an “All Banks Precharge” command must be issued.
1	0	1	The SDRAM Controller issues an extended load mode register command when the SDRAM device is accessed regardless of the cycle. The address offset with respect to the SDRAM device base address is used to program the Mode Register. For instance, when this mode is activated, an access to the “SDRAM_Base + offset” address generates an “Extended Load Mode Register” command with the value “offset” written to the SDRAM device Mode Register.
1	1	0	Deep power-down mode. Enters deep power-down mode.

17.6.2 SDRAMC Refresh Timer Register

Register Name: SDRAMC_TR

Access Type: Read/Write

Reset Value: 0x00000000

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

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

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

7	6	5	4	3	2	1	0
COUNT							

- **COUNT: SDRAMC Refresh Timer Count**

This 12-bit field is loaded into a timer that generates the refresh pulse. Each time the refresh pulse is generated, a refresh burst is initiated. The value to be loaded depends on the SDRAMC clock frequency (MCK: Master Clock), the refresh rate of the SDRAM device and the refresh burst length where 15.6 µs per row is a typical value for a burst of length one.

To refresh the SDRAM device, this 12-bit field must be written. If this condition is not satisfied, no refresh command is issued and no refresh of the SDRAM device is carried out.

17.6.3 SDRAMC Configuration Register

Register Name: SDRAMC_CR

Access Type: Read/Write

Reset Value: 0x852372C0

31	30	29	28	27	26	25	24
TXSR				TRAS			

23	22	21	20	19	18	17	16
TRCD				TRP			

15	14	13	12	11	10	9	8
TRC				TWR			

7	6	5	4	3	2	1	0
DBW	CAS		NB	NR		NC	

- **NC: Number of Column Bits**

Reset value is 8 column bits.

NC		Column Bits
0	0	8
0	1	9
1	0	10
1	1	11

- **NR: Number of Row Bits**

Reset value is 11 row bits.

NR		Row Bits
0	0	11
0	1	12
1	0	13
1	1	Reserved

- **NB: Number of Banks**

Reset value is two banks.

NB	Number of Banks
0	2
1	4

- **CAS: CAS Latency**

Reset value is two cycles.

In the SDRAMC, only a CAS latency of one, two and three cycles are managed. In any case, another value must be programmed.

CAS		CAS Latency (Cycles)
0	0	Reserved
0	1	1
1	0	2
1	1	3

- **DBW: Data Bus Width**

Reset value is 16 bits

0: Data bus width is 32 bits.

1: Data bus width is 16 bits.

- **TWR: Write Recovery Delay**

Reset value is two cycles.

This field defines the Write Recovery Time in number of cycles. Number of cycles is between 0 and 15.

- **TRC: Row Cycle Delay**

Reset value is seven cycles.

This field defines the delay between a Refresh and an Activate Command in number of cycles. Number of cycles is between 0 and 15.

- **TRP: Row Precharge Delay**

Reset value is three cycles.

This field defines the delay between a Precharge Command and another Command in number of cycles. Number of cycles is between 0 and 15.

- **TRCD: Row to Column Delay**

Reset value is two cycles.

This field defines the delay between an Activate Command and a Read/Write Command in number of cycles. Number of cycles is between 0 and 15.

- **TRAS: Active to Precharge Delay**

Reset value is five cycles.

This field defines the delay between an Activate Command and a Precharge Command in number of cycles. Number of cycles is between 0 and 15.

- **TXSR: Exit Self Refresh to Active Delay**

Reset value is eight cycles.

This field defines the delay between SCKE set high and an Activate Command in number of cycles. Number of cycles is between 0 and 15.

17.6.3.1 SDRAMC High Speed Register

Register Name: SDRAMC_HSR

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**
–	–	–	–	–	–	–	DA

- **DA: Decode Cycle Enable**

A decode cycle can be added on the addresses as soon as a non-sequential access is performed on the AHB bus.

The addition of the decode cycle allows the SDRAMC to gain time to access the SDRAM memory.

0: Decode cycle is disabled.

1: Decode cycle is enabled.

17.6.4 SDRAMC Low Power Register

Register Name: SDRAMC_LPR

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
–	–	TIMEOUT		DS		TCSR	

7	6	5	4	3	2	1	0
–	PASR			–	–	LPCB	

- **LPCB: Low-power Configuration Bits**

00	Low Power Feature is inhibited: no Power-down, Self-refresh or Deep Power-down command is issued to the SDRAM device.
01	The SDRAM Controller issues a Self-refresh command to the SDRAM device, the SDCLK clock is deactivated and the SDCKE signal is set low. The SDRAM device leaves the Self Refresh Mode when accessed and enters it after the access.
10	The SDRAM Controller issues a Power-down Command to the SDRAM device after each access, the SDCKE signal is set to low. The SDRAM device leaves the Power-down Mode when accessed and enters it after the access.
11	The SDRAM Controller issues a Deep Power-down command to the SDRAM device. This mode is unique to low-power SDRAM.

- **PASR: Partial Array Self-refresh (only for low-power SDRAM)**

PASR parameter is transmitted to the SDRAM during initialization to specify whether only one quarter, one half or all banks of the SDRAM array are enabled. Disabled banks are not refreshed in self-refresh mode. This parameter must be set according to the SDRAM device specification.

After initialization, as soon as PASR field is modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and PASR bits are updated before entry in self-refresh mode.

- **TCSR: Temperature Compensated Self-Refresh (only for low-power SDRAM)**

TCSR parameter is transmitted to the SDRAM during initialization to set the refresh interval during self-refresh mode depending on the temperature of the low-power SDRAM. This parameter must be set according to the SDRAM device specification.

After initialization, as soon as TCSR field is modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and TCSR bits are updated before entry in self-refresh mode.

- **DS: Drive Strength (only for low-power SDRAM)**

DS parameter is transmitted to the SDRAM during initialization to select the SDRAM strength of data output. This parameter must be set according to the SDRAM device specification.

After initialization, as soon as DS field is modified and self-refresh mode is activated, the Extended Mode Register is accessed automatically and DS bits are updated before entry in self-refresh mode.

- **TIMEOUT: Time to define when low-power mode is enabled**

00	The SDRAM controller activates the SDRAM low-power mode immediately after the end of the last transfer.
01	The SDRAM controller activates the SDRAM low-power mode 64 clock cycles after the end of the last transfer.
10	The SDRAM controller activates the SDRAM low-power mode 128 clock cycles after the end of the last transfer.
11	Reserved.

17.6.5 SDRAMC Interrupt Enable Register

Register Name: SDRAMC_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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: No effect.

1: Enables the refresh error interrupt.

17.6.6 SDRAMC Interrupt Disable Register

Register Name: SDRAMC_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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: No effect.

1: Disables the refresh error interrupt.

17.6.7 SDRAMC Interrupt Mask Register

Register Name: SDRAMC_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
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: The refresh error interrupt is disabled.

1: The refresh error interrupt is enabled.

17.6.8 SDRAMC Interrupt Status Register

Register Name: SDRAMC_ISR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	RES

- **RES: Refresh Error Status**

0: No refresh error has been detected since the register was last read.

1: A refresh error has been detected since the register was last read.

17.6.9 SDRAMC Memory Device Register

Register Name: SDRAMC_MDR

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
–	–	–	–	–	–	MD	

- **MD: Memory Device Type**

00	SDRAM
01	Low-power SDRAM
10	Reserved
11	Reserved.

18. Peripheral DMA Controller (PDC)

18.1 Description

The Peripheral DMA Controller (PDC) transfers data between on-chip serial peripherals and the on- and/or off-chip memories. The link between the PDC and a serial peripheral is operated by the AHB to ABP bridge.

The PDC contains 23 channels. The full-duplex peripherals feature 20 mono directional channels used in pairs (transmit only or receive only). The half-duplex peripherals feature 3 bi-directional channels.

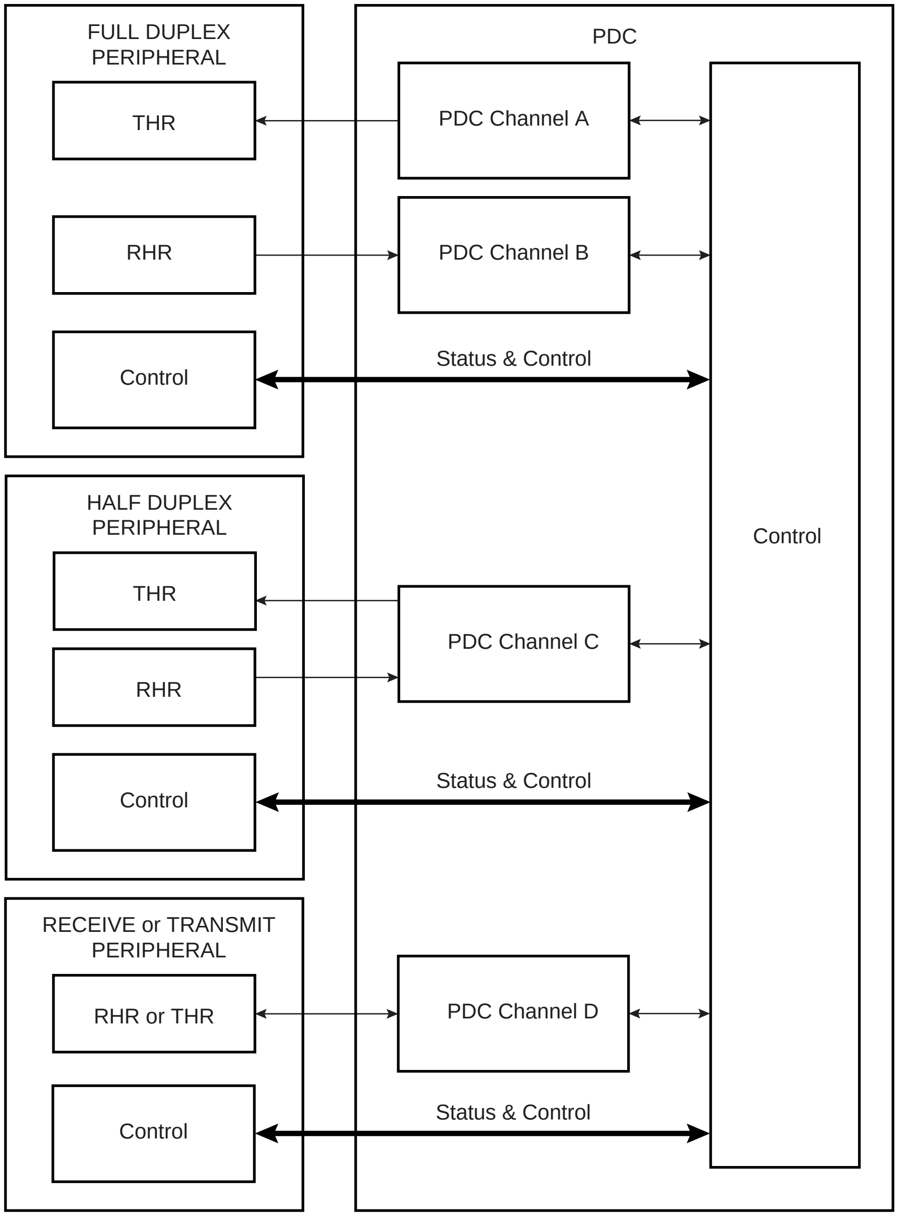
The user interface of each PDC channel is integrated into the user interface of the peripheral it serves. The user interface of mono directional channels (receive only or transmit only), contains two 32-bit memory pointers and two 16-bit counters, one set (pointer, counter) for current transfer and one set (pointer, counter) for next transfer. The bi-directional channel user interface contains four 32-bit memory pointers and four 16-bit counters. Each set (pointer, counter) is used by current transmit, next transmit, current receive and next receive.

Using the PDC removes processor overhead by reducing its intervention during the transfer. This significantly reduces the number of clock cycles required for a data transfer, which improves microcontroller performance.

To launch a transfer, the peripheral triggers its associated PDC channels by using transmit and receive signals. When the programmed data is transferred, an end of transfer interrupt is generated by the peripheral itself.

18.2 Block Diagram

Figure 18-1. Block Diagram



18.3 Functional Description

18.3.1 Configuration

The PDC channel user interface enables the user to configure and control data transfers for each channel. The user interface of each PDC channel is integrated into the associated peripheral user interface.

The user interface of a serial peripheral, whether it is full or half duplex, contains four 32-bit pointers (RPR, RNPR, TPR, TNPR) and four 16-bit counter registers (RCR, RNCR, TCR, TNCR). However, the transmit and receive parts of each type are programmed differently: the transmit and receive parts of a full duplex peripheral can be programmed at the same time, whereas only one part (transmit or receive) of a half duplex peripheral can be programmed at a time.

32-bit pointers define the access location in memory for current and next transfer, whether it is for read (transmit) or write (receive). 16-bit counters define the size of current and next transfers. It is possible, at any moment, to read the number of transfers left for each channel.

The PDC has dedicated status registers which indicate if the transfer is enabled or disabled for each channel. The status for each channel is located in the associated peripheral status register. Transfers can be enabled and/or disabled by setting TXTEN/TXTDIS and RXTEN/RXTDIS in the peripheral's Transfer Control Register.

At the end of a transfer, the PDC channel sends status flags to its associated peripheral. These flags are visible in the peripheral status register (ENDRX, ENDTX, RXBUFF, and TXBUFE). Refer to Section 18.3.3 and to the associated peripheral user interface.

18.3.2 Memory Pointers

Each full duplex peripheral is connected to the PDC by a receive channel and a transmit channel. Both channels have 32-bit memory pointers that point respectively to a receive area and to a transmit area in on- and/or off-chip memory.

Each half duplex peripheral is connected to the PDC by a bidirectional channel. This channel has two 32-bit memory pointers, one for current transfer and the other for next transfer. These pointers point to transmit or receive data depending on the operating mode of the peripheral.

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

If a memory pointer address changes in the middle of a transfer, the PDC channel continues operating using the new address.

18.3.3 Transfer Counters

Each channel has two 16-bit counters, one for current transfer and the other one for next transfer. These counters define the size of data to be transferred by the channel. The current transfer counter is decremented first as the data addressed by current memory pointer starts to be transferred. When the current transfer counter reaches zero, the channel checks its next transfer counter. If the value of next counter is zero, the channel stops transferring data and sets the appropriate flag. But if the next counter value is greater then zero, the values of the next pointer/next counter are copied into the current pointer/current counter and the channel resumes the transfer whereas next pointer/next counter get zero/zero as values. At the end of this transfer the PDC channel sets the appropriate flags in the Peripheral Status Register.

The following list gives an overview of how status register flags behave depending on the counters' values:

- ENDRX flag is set when the PERIPH_RCR register reaches zero.
- RXBUFF flag is set when both PERIPH_RCR and PERIPH_RNCR reach zero.
- ENDTX flag is set when the PERIPH_TCR register reaches zero.
- TXBUFE flag is set when both PERIPH_TCR and PERIPH_TNCR reach zero.

These status flags are described in the Peripheral Status Register.

18.3.4 Data Transfers

The serial peripheral triggers its associated PDC channels' transfers using transmit enable (TXEN) and receive enable (RXEN) flags in the transfer control register integrated in the peripheral's user interface.

When the peripheral receives an external data, it sends a Receive Ready signal to its PDC receive channel which then requests access to the Matrix. When access is granted, the PDC receive channel starts reading the peripheral Receive Holding Register (RHR). The read data are stored in an internal buffer and then written to memory.

When the peripheral is about to send data, it sends a Transmit Ready to its PDC transmit channel which then requests access to the Matrix. When access is granted, the PDC transmit channel reads data from memory and puts them to Transmit Holding Register (THR) of its associated peripheral. The same peripheral sends data according to its mechanism.

18.3.5 PDC Flags and Peripheral Status Register

Each peripheral connected to the PDC sends out receive ready and transmit ready flags and the PDC sends back flags to the peripheral. All these flags are only visible in the Peripheral Status Register.

Depending on the type of peripheral, half or full duplex, the flags belong to either one single channel or two different channels.

18.3.5.1 Receive Transfer End

This flag is set when PERIPH_RCR register reaches zero and the last data has been transferred to memory.

It is reset by writing a non zero value in PERIPH_RCR or PERIPH_RNCR.

18.3.5.2 Transmit Transfer End

This flag is set when PERIPH_TCR register reaches zero and the last data has been written into peripheral THR.

It is reset by writing a non zero value in PERIPH_TCR or PERIPH_TNCR.

18.3.5.3 Receive Buffer Full

This flag is set when PERIPH_RCR register reaches zero with PERIPH_RNCR also set to zero and the last data has been transferred to memory.

It is reset by writing a non zero value in PERIPH_TCR or PERIPH_TNCR.

18.3.5.4 Transmit Buffer Empty

This flag is set when PERIPH_TCR register reaches zero with PERIPH_TNCR also set to zero and the last data has been written into peripheral THR.

It is reset by writing a non zero value in PERIPH_TCR or PERIPH_TNCR.

18.4 Peripheral DMA Controller (PDC) User Interface

Table 18-1. Memory Map

Offset	Register	Name	Access	Reset State
0x100	Receive Pointer Register	PERIPH[(1)]_RPR	Read/Write	0
0x104	Receive Counter Register	PERIPH_RCR	Read/Write	0
0x108	Transmit Pointer Register	PERIPH_TPR	Read/Write	0
0x10C	Transmit Counter Register	PERIPH_TCR	Read/Write	0
0x110	Receive Next Pointer Register	PERIPH_RNPR	Read/Write	0
0x114	Receive Next Counter Register	PERIPH_RNCR	Read/Write	0
0x118	Transmit Next Pointer Register	PERIPH_TNPR	Read/Write	0
0x11C	Transmit Next Counter Register	PERIPH_TNCR	Read/Write	0
0x120	Transfer Control Register	PERIPH_PTCR	Write	0
0x124	Transfer Status Register	PERIPH_PTSR	Read	0

Note: 1. PERIPH: Ten registers are mapped in the peripheral memory space at the same offset. These can be defined by the user according to the function and the peripheral desired (DBGU, USART, SSC, SPI, MCI, etc.)

18.4.1 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 Register**

RXPTR must be set to receive buffer address.

When a half duplex peripheral is connected to the PDC, RXPTR = TXPTR.

18.4.2 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 Register**

RXCTR must be set to receive buffer size.

When a half duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0 = Stops peripheral data transfer to the receiver

1 - 65535 = Starts peripheral data transfer if corresponding channel is active

18.4.3 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 Counter Register**

TXPTR must be set to transmit buffer address.

When a half duplex peripheral is connected to the PDC, RXPTR = TXPTR.

18.4.4 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 Register**

TXCTR must be set to transmit buffer size.

When a half duplex peripheral is connected to the PDC, RXCTR = TXCTR.

0 = Stops peripheral data transfer to the transmitter

1- 65535 = Starts peripheral data transfer if corresponding channel is active

18.4.5 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**

RXNPTR contains next receive buffer address.

When a half duplex peripheral is connected to the PDC, RXNPTR = TXNPTR.

18.4.6 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**
RXNCTR							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RXNCTR							

- **RXNCTR: Receive Next Counter**

RXNCTR contains next receive buffer size.

When a half duplex peripheral is connected to the PDC, RXNCTR = TXNCTR.

18.4.7 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**

TXNPTR contains next transmit buffer address.

When a half duplex peripheral is connected to the PDC, RXNPTR = TXNPTR.

18.4.8 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**
TXNCTR							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TXNCTR							

- **TXNCTR: Transmit Counter Next**

TXNCTR contains next transmit buffer size.

When a half duplex peripheral is connected to the PDC, RXNCTR = TXNCTR.

18.4.9 Transfer Control Register

Register Name: PERIPH_PTCR

Access Type: 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**
–	–	–	–	–	–	TXTDIS	TXTEN
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
–	–	–	–	–	–	RXTDIS	RXTEN

- **RXTEN: Receiver Transfer Enable**

0 = No effect.

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

When a half duplex peripheral is connected to the PDC, enabling the receiver channel requests automatically disables the transmitter channel requests. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **RXTDIS: Receiver Transfer Disable**

0 = No effect.

1 = Disables the PDC receiver channel requests.

When a half duplex peripheral is connected to the PDC, disabling the receiver channel requests also disables the transmitter channel requests.

- **TXTEN: Transmitter Transfer Enable**

0 = No effect.

1 = Enables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, it enables the transmitter channel requests only if RXTEN is not set. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **TXTDIS: Transmitter Transfer Disable**

0 = No effect.

1 = Disables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, disabling the transmitter channel requests disables the receiver channel requests.

18.4.10 Transfer Status Register

Register Name: PERIPH_PTSR

Access Type: Read

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

- **RXTEN: Receiver Transfer Enable**

0 = PDC Receiver channel requests are disabled.

1 = PDC Receiver channel requests are enabled.

- **TXTEN: Transmitter Transfer Enable**

0 = PDC Transmitter channel requests are disabled.

1 = PDC Transmitter channel requests are enabled

19. Clock Generator

19.1 Description

The Clock Generator is made up of 2 PLL, a Main Oscillator, and a 32,768 Hz low-power Oscillator.

It provides the following clocks:

- SLCK, the Slow Clock, which is the only permanent clock within the system
- MAINCK is the output of the Main Oscillator

The Clock Generator User Interface is embedded within the Power Management Controller one and is described in Section 20.9. However, the Clock Generator registers are named CKGR_.

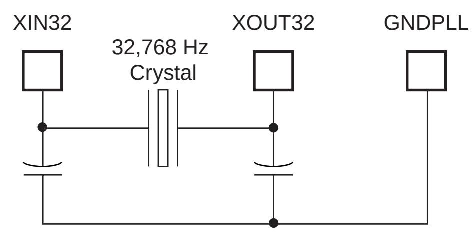
- PLLACK is the output of the Divider and PLL A block
- PLLBCK is the output of the Divider and PLL B block

19.2 Slow Clock Crystal Oscillator

The Clock Generator integrates a 32,768 Hz low-power oscillator. The X32IN and X32OUT pins must be connected to a 32,768 Hz crystal. Two external capacitors must be wired as shown in Figure 19-1.

X32EN input allows embedded slow clock oscillator bypass when pulled down, so that an external clock line (up to 50 MHz) can be directly connected to X32IN.

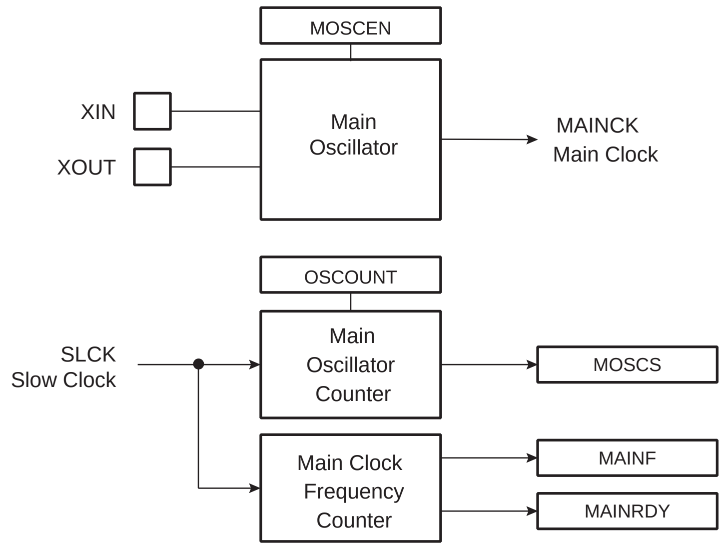
Figure 19-1. Typical Slow Clock Crystal Oscillator Connection



19.3 Main Oscillator

Figure 19-2 shows the Main Oscillator block diagram.

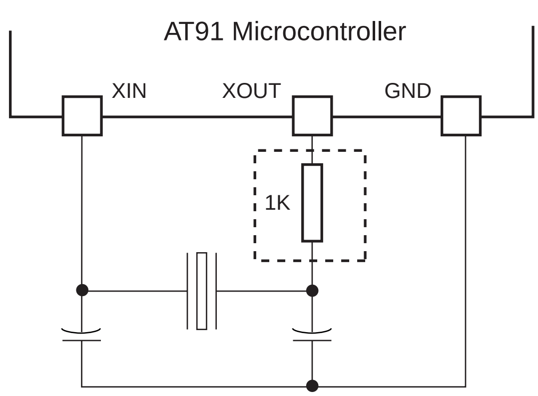
Figure 19-2. Main Oscillator Block Diagram



19.3.1 Main Oscillator Connections

The Clock Generator integrates a Main Oscillator that is designed for a 8 to 16 MHz fundamental crystal. In application where are used USB Host or Device peripherals, the Main Oscillator frequency must be 12 MHz because PLLB provides USB working frequency (96/48 MHz) through a multiplication (x8) of its clock source. The typical crystal connection is illustrated in Figure 19-3. For further details on the electrical characteristics of the Main Oscillator, see the section "DC Characteristics" of the product datasheet.

Figure 19-3. Typical Crystal Connection



19.3.2 Main Oscillator Startup Time

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

19.3.3 Main Oscillator Control

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

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

When disabling the main oscillator by clearing the MOSCEN bit in CKGR_MOR, the MOSCS bit in PMC_SR is automatically cleared, indicating the main clock is off.

When enabling the main oscillator, the user must initiate the main oscillator counter with a value corresponding to the startup time of the oscillator. This startup time depends on the crystal frequency connected to the main oscillator.

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

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

19.3.4 Main Clock Frequency Counter

The Main Oscillator features a Main Clock frequency counter that provides the quartz frequency connected to the Main Oscillator. Generally, this value is known by the system designer; however, it could be useful for some application program determine Main Oscillator frequency.

The Main Clock frequency counter starts incrementing at the Main Clock speed after the next rising edge of the Slow Clock as soon as the Main Oscillator is stable, i.e., as soon as the MOSCS bit is set. Then, at the 16th falling edge of Slow Clock, the MAINRDY bit in CKGR_MCFR (Main Clock Frequency Register) is set and the counter stops counting. Its value can be read in the MAINF field of CKGR_MCFR and gives the number of Main Clock cycles during 16 periods of Slow Clock, so that the frequency of the crystal connected on the Main Oscillator can be determined.

19.3.5 Main Oscillator Bypass

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

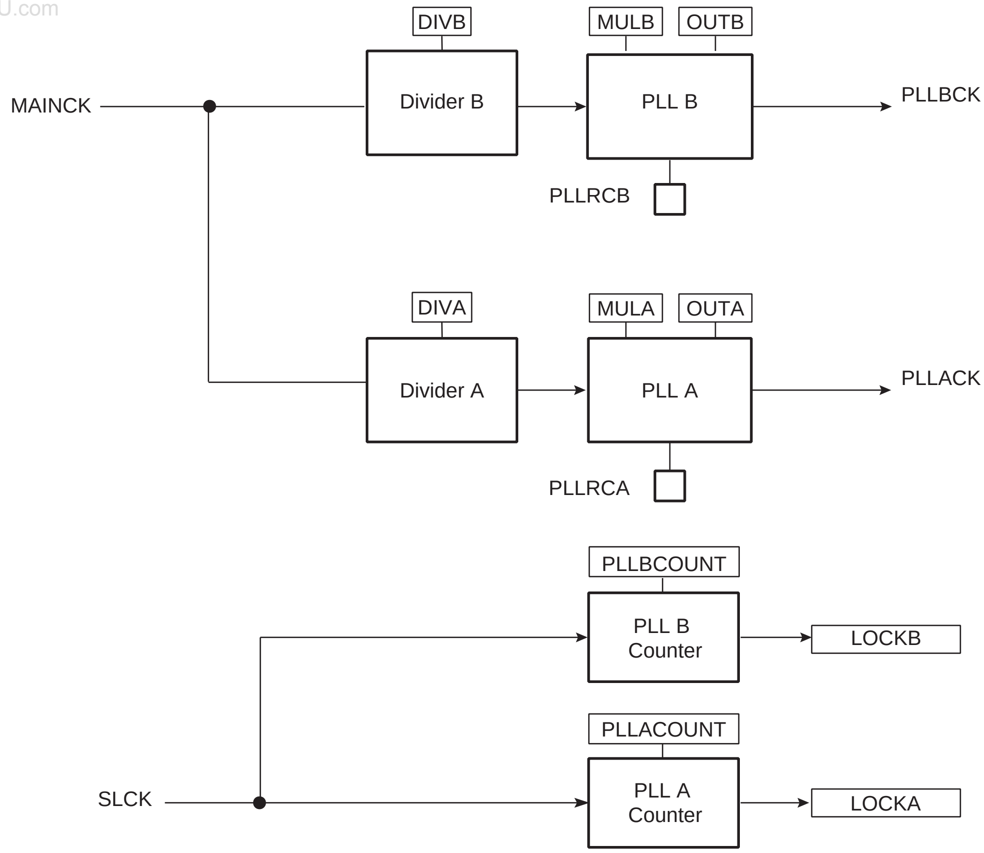
19.4 Divider and PLL Block

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

On the contrary PLLB provides fixed MULB (x8) and DIVB (1) factors despite of related register values. Anyway these registers must be properly programmed to let the PLLB be enabled and also to be SW compatible with all D940HF revisions.

Figure 19-4 shows the block diagram of the divider and PLL blocks.

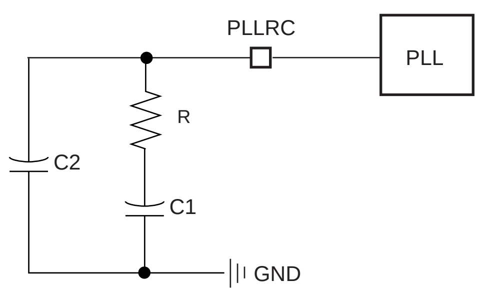
Figure 19-4. Divider and PLL Block Diagram



19.4.1 PLL Filter

The PLL requires connection to an external second-order filter through the PLL_RCA and/or PLL_RCB pin. Figure 19-5 shows a schematic of these filters.

Figure 19-5. PLL Capacitors and Resistors



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

19.4.2 Divider and Phase Lock Loop Programming

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

The PLL allows multiplication of the divider's outputs. The PLL clock signal has a frequency that depends on the respective source signal frequency and on the parameters DIV and MUL. The factor applied to the source signal frequency is (MUL + 1)/DIV. When MUL is written to 0, the corresponding PLL is disabled and its power consumption is saved. Re-enabling the PLL can be performed by writing a value higher than 0 in the MUL field.

Whenever the PLL is re-enabled or one of its parameters is changed, the LOCK bit (LOCKA or LOCKB) in PMC_SR is automatically cleared. The values written in the PLLCOUNT field (PLLA-COUNT or PLLBCOUNT) in CKGR_PLLR (CKGR_PLLAR or CKGR_PLLBR), are loaded in the PLL counter. The PLL counter then decrements at the speed of the Slow Clock until it reaches 0. At this time, the LOCK bit is set in PMC_SR and can trigger an interrupt to the processor. The user has to load the number of Slow Clock cycles required to cover the PLL transient time into the PLLCOUNT field. The transient time depends on the PLL filter. The initial state of the PLL and its target frequency can be calculated using a specific tool provided by Atmel.

20. Power Management Controller (PMC)

20.1 Description

The Power Management Controller (PMC) optimizes power consumption by controlling all system and user peripheral clocks. The PMC enables/disables the clock inputs to many of the peripherals and the ARM Processor.

The Power Management Controller provides the following clocks:

- MCK, the Master Clock, programmable from a few hundred Hz to the maximum operating frequency of the device. It is available to the modules running permanently, such as the AIC and the Memory Controller.
- Processor Clock (PCK), must be switched off when entering processor in Idle Mode.
- Peripheral Clocks, typically MCK, provided to the embedded peripherals (USART, SSC, SPI, TWI, TC, MCI, etc.) and independently controllable. In order to reduce the number of clock names in a product, the Peripheral Clocks are named MCK in the product datasheet.
- UHP Clock (UHPCK), required by USB Host Port operations.
- Programmable Clock Outputs can be selected from the clocks provided by the clock generator and driven on the PCKx pins.

20.2 Master Clock Controller

The Master Clock Controller provides selection and division of the Master Clock (MCK). MCK is the clock provided to all the peripherals and the memory controller.

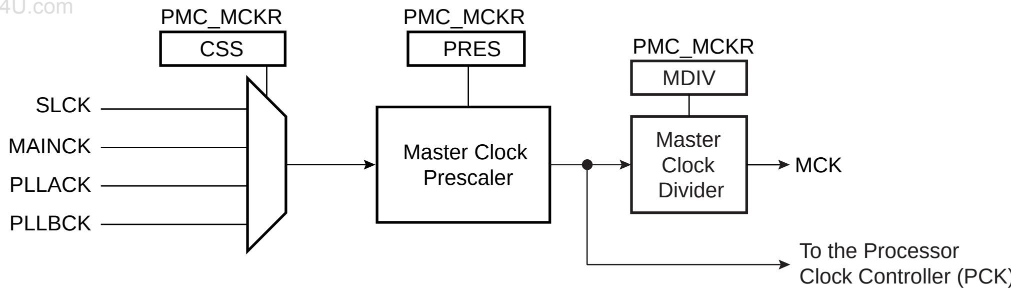
The Master Clock is selected from one of the clocks provided by the Clock Generator. Selecting the Slow Clock provides a Slow Clock signal to the whole device. Selecting the Main Clock saves power consumption of the PLLs.

The Master Clock Controller is made up of a clock selector and a prescaler. It also contains a Master Clock divider which 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. The Master Clock divider can be programmed 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.

Figure 20-1. Master Clock Controller



20.3 Processor Clock Controller

The PMC features a Processor Clock Controller (PCK) that implements the Processor Idle Mode. The Processor Clock can be disabled by writing the System Clock Disable Register (PMC_SCDR). The status of this clock (at least for debug purposes) can be read in the System Clock Status Register (PMC_SCSR).

The Processor Clock PCK is enabled after a reset and is automatically re-enabled by any enabled interrupt. The Processor Idle Mode is achieved by disabling the Processor Clock and entering Wait for Interrupt Mode. The Processor Clock is automatically re-enabled by any enabled fast or normal interrupt, or by the reset of the product.

Note: The ARM Wait for Interrupt mode is entered with CP15 coprocessor operation. Refer to the Atmel application note, Optimizing Power Consumption of AT91SAM9261-based Systems, lit. number 6217.

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.

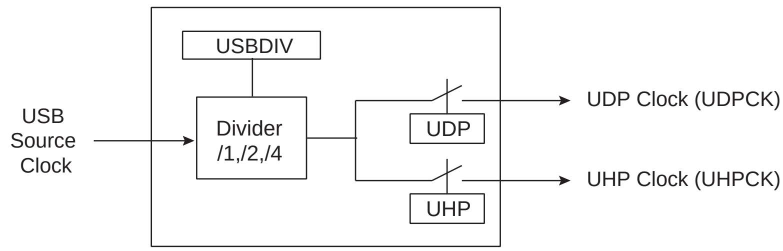
20.4 USB Clock Controller

The USB Source Clock is always generated from the PLL B output. If using the USB, the user must program the PLL to generate a 96 MHz starting from a 12 MHz source on the USBDIV bit in CKGR_PLLBR (see Figure 20-2), even if the PLLB has fixed MUL/DIV factors that don't depend on the programmed values. Anyway if these registers are not programmed the PLLB cannot be enabled.

When the PLL B output is stable, i.e., the LOCKB is set:

- The USB host clock can be enabled by setting the UHP bit in PMC_SCER. To save power on this peripheral when it is not used, the user can set the UHP bit in PMC_SCDR. The UHP bit in PMC_SCSR gives the activity of this clock. The USB host port require both the 12/48 MHz signal and the Master Clock. The Master Clock may be controlled via the Master Clock Controller.

Figure 20-2. USB Clock Controller



20.5 Peripheral Clock Controller

The Power Management Controller controls the clocks of each embedded peripheral by the way of the Peripheral Clock Controller. The user can individually enable and disable the Master Clock on the peripherals by writing into the Peripheral Clock Enable (PMC_PCER) and Peripheral Clock Disable (PMC_PCDR) registers. The status of the peripheral clock activity can be read in the Peripheral Clock Status Register (PMC_PCSR).

When a peripheral clock is disabled, the clock is immediately stopped. The peripheral clocks are automatically disabled after a reset.

In order to stop a peripheral, it is recommended that the system software wait until the peripheral has executed its last programmed operation before disabling the clock. This is to avoid data corruption or erroneous behavior of the system.

The bit number within the Peripheral Clock Control registers (PMC_PCER, PMC_PCDR, and PMC_PCSR) is the Peripheral Identifier defined at the product level. Generally, the bit number corresponds to the interrupt source number assigned to the peripheral.

20.6 Programmable Clock Output Controller

The PMC controls 5 signals to be output on PCKx lines: PCK0 to PCK3 are connected to external pins (via PIOS), while PCK4 is used to provide the working frequency to MagicV memories (2x AHB system clock). Each signal can be independently programmed via the PMC_PCKx registers.

PCKx can be independently selected between the Slow clock, the PLL A output, the PLL B output and the main clock by writing the CSS field in PMC_PCKx. Each output signal can also be divided by a power of 2 between 1 and 64 by writing the PRES (Prescaler) field in PMC_PCKx.

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

Moreover, like the PCK, a status bitin 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.

20.7 Programming Sequence

1. Enabling the Main Oscillator:

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

Once this register has been correctly configured, the user must wait for MOSCS field in the PMC_SR register to be set. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to MOSCS has been enabled in the PMC_IER register.

Code Example:

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

Start Up Time = 8 * OSCOUNT / SLCK = 56 Slow Clock Cycles.

So, the main oscillator will be enabled (MOSCS bit set) after 56 Slow Clock Cycles.

2. Checking the Main Oscillator Frequency (Optional):

In some situations the user may need an accurate measure of the main oscillator frequency. This measure can be accomplished via the CKGR_MCFR register.

Once the MAINRDY field is set in CKGR_MCFR register, the user may read the MAINF field in CKGR_MCFR register. This provides the number of main clock cycles within sixteen slow clock cycles.

3. Setting PLL A and divider A:

All parameters necessary to configure PLL A and divider A are located in the CKGR_PLLAR register.

It is important to note that Bit 29 must always be set to 1 when programming the CKGR_PLLAR register.

The DIVA field is used to control the divider A itself. The user can program a value between 0 and 255. Divider A output is divider A input divided by DIVA. By default, DIVA parameter is set to 0 which means that divider A is turned off.

The OUTA field is used to select the PLL A output frequency range.

The MULA field is the PLL A multiplier factor. This parameter can be programmed between 0 and 2047. If MULA is set to 0, PLL A will be turned off. Otherwise PLL A output frequency is PLL A input frequency multiplied by (MULA + 1).

The PLLACOUNT field specifies the number of slow clock cycles before LOCKA bit is set in the PMC_SR register after CKGR_PLLAR register has been written.

Once CKGR_PLLAR register has been written, the user is obliged to wait for the LOCKA bit to be set in the PMC_SR register. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to LOCKA has been enabled in the PMC_IER register.

All parameters in CKGR_PLLAR can be programmed in a single write operation. If at some stage one of the following parameters, SRCA, MULA, DIVA is modified, LOCKA bit will go low to indicate that PLL A is not ready yet. When PLL A is locked, LOCKA will be set again. User has to wait for LOCKA bit to be set before using the PLL A output clock.

Code Example:

```
write_register(CKGR_PLLAR,0x20030605)
```

PLL A and divider A are enabled. PLL A input clock is main clock divided by 5. PLL An output clock is PLL A input clock multiplied by 4. Once CKGR_PLLAR has been written, LOCKA bit will be set after six slow clock cycles.

4. Setting PLL B and divider B:

All parameters needed to configure PLL B and divider B are located in the CKGR_PLLBR register.

PLLB has fixed MUL/DIV factors that are independent from MULB/DIVB fields but they must be programmed anyway to let the PLLB be enabled.

The PLLBCOUNT field specifies the number of slow clock cycles before LOCKB bit is set in the PMC_SR register after CKGR_PLLBR register has been written.

Once the PMC_PLLB register has been written, the user must wait for the LOCKB bit to be set in the PMC_SR register. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to LOCKB has been enabled in the PMC_IER register. All parameters in CKGR_PLLBR can be programmed in a single write operation. The user is constrained to wait for LOCKB bit to be set before using the PLL A output clock.

The USBDIV field is used to control the additional divider by 1, 2 or 4, which generates the USB clock(s): since the fixed MUL/DIV factors provides a x8 multiplication of 12 MHz source clock, to get the 48 MHz USB working frequency the USBDIV must be set to 2.

5. Selection of Master Clock and Processor Clock

The Master Clock and the Processor Clock are configurable via the PMC_MCKR register.

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

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

Master Clock Prescaler and PCK4 (MagicV memories clock) must be the same.

The MDIV field is used to control the Master Clock prescaler. It is possible to choose between different values (0, 1, 2). The Master Clock output is ARM Processor Clock divided by 1, 2 or 4, depending on the value programmed in MDIV. By default, MDIV is set to 0, which indicates that the Processor Clock is equal to the Master Clock.

Master Clock Divider must be 1 to get finally: ARM clock = MagicV memories clock = 2x AHB system clock = 2x MagicV clock.

Once the PMC_MCKR register has been written, the user must wait for the MCKRDY bit to be set in the PMC_SR register. This can be done either by polling the status register or by waiting for the interrupt line to be raised if the associated interrupt to MCKRDY has been enabled in the PMC_IER register.

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

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

If at some stage one of the following parameters, CSS or PRES, is modified, the MCKRDY bit will go low to indicate that the Master Clock and the Processor Clock are not ready yet. The user must wait for MCKRDY bit to be set again before using the Master and Processor Clocks.

Note: IF PLLx clock was selected as the Master Clock and the user decides to modify it by writing in CKGR_PLLR (CKGR_PLLAR or CKGR_PLLBR), the MCKRDY flag will go low while PLL is unlocked. Once PLL is locked again, LOCK (LOCKA or LOCKB) goes high and MCKRDY is set. While PLLA is unlocked, the Master Clock selection is automatically changed to Slow Clock. While PLLB is unlocked, the Master Clock selection is automatically changed to Main Clock. For further information, see Section 20.8.2. "Clock Switching Waveforms" on page 244.

Code Example:

```
write_register(PMC_MCKR,0x00000001)
wait (MCKRDY=1)

write_register(PMC_MCKR,0x00000011)
wait (MCKRDY=1)
```

The Master Clock is main clock divided by 16.

The Processor Clock is the Master Clock.

6. Selection of Programmable clocks

Programmable clocks are controlled via registers; PMC_SCER, PMC_SCDR and PMC_SCSR.

Programmable clocks can be enabled and/or disabled via the PMC_SCER and PMC_SCDR registers. Depending on the system used, 5 Programmable clocks can be enabled or disabled. The PMC_SCSR provides a clear indication as to which Programmable clock is enabled. By default all Programmable clocks are disabled.

PMC_PCKx registers are used to configure Programmable clocks.

The CSS field is used to select the Programmable clock divider source. Four clock options are available: main clock, slow clock, PLLACK, PLLBCK. By default, the clock source selected is slow clock.

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

Once the PMC_PCKx register has been programmed, The corresponding Programmable clock must be enabled and the user is constrained to wait for the PCKRDYx bit to be set in the PMC_SR register. This can be done either by polling the status register or by waiting the interrupt line to be raised if the associated interrupt to PCKRDYx has been enabled in the PMC_IER register. All parameters in PMC_PCKx can be programmed in a single write operation.

If the CSS and PRES parameters are to be modified, the corresponding Programmable clock must be disabled first. The parameters can then be modified. Once this has been done, the user must re-enable the Programmable clock and wait for the PCKRDYx bit to be set.

Code Example:

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

Programmable clock 0 is main clock divided by 32.

7. Enabling Peripheral Clocks

Once all of the previous steps have been completed, the peripheral clocks can be enabled and/or disabled via registers PMC_PCER and PMC_PCDR.

Depending on the system used, 22 peripheral clocks can be enabled or disabled. The PMC_PCSR provides a clear view as to which peripheral clock is enabled.

Note: Each enabled peripheral clock corresponds to Master Clock.

Code Examples:

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

Peripheral clocks 4 and 8 are enabled.

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

Peripheral clock 4 is disabled.

20.8 Clock Switching Details

20.8.1 Master Clock Switching Timings

Table 20-1 and Table 20-2 give the worst case timings required for the Master Clock to switch from one selected clock to another one. This is in the event that the prescaler is de-activated. When the prescaler is activated, an additional time of 64 clock cycles of the new selected clock has to be added.

Table 20-1. Clock Switching Timings (Worst Case)

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

Notes: 1. PLL designates either the PLL A or the PLL B Clock.
2. PLLCOUNT designates either PLLACOUNT or PLLBCOUNT.

Table 20-2. Clock Switching Timings Between Two PLLs (Worst Case)

From / To	PLLA Clock	PLLB Clock
PLLA Clock	2.5 x PLLA Clock + 4 x SLCK + PLLACOUNT x SLCK	3 x PLLA Clock + 4 x SLCK + 1.5 x PLLA Clock
PLLB Clock	3 x PLLB Clock + 4 x SLCK + 1.5 x PLLB Clock	2.5 x PLLB Clock + 4 x SLCK + PLLBCOUNT x SLCK

20.8.2 Clock Switching Waveforms

Figure 20-3. Switch Master Clock from Slow Clock to PLL Clock

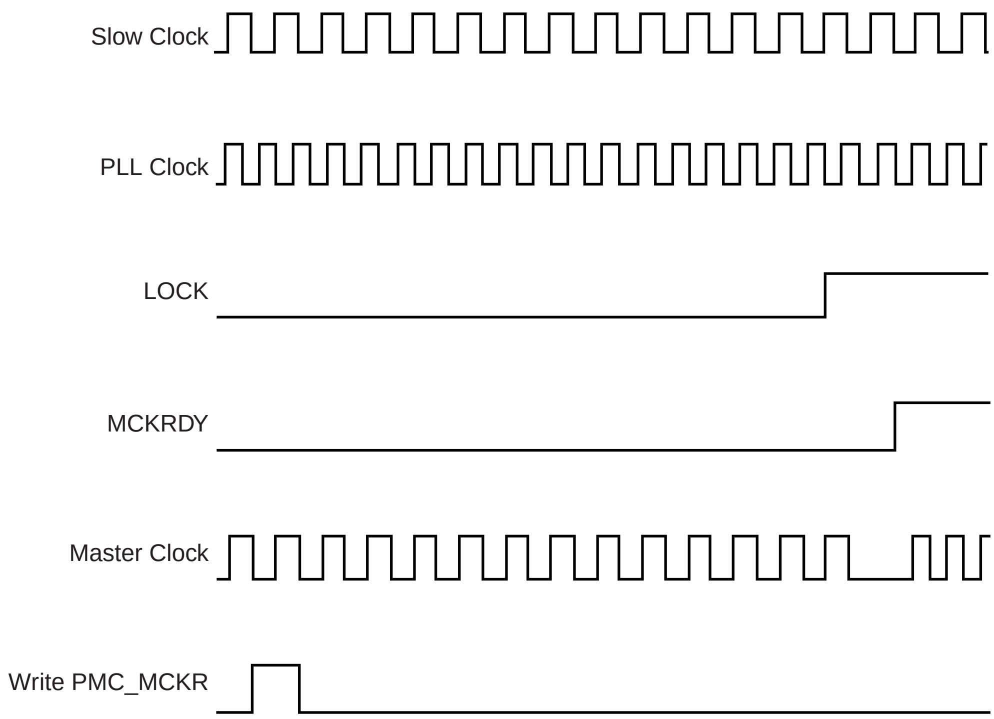


Figure 20-4. Switch Master Clock from Main Clock to Slow Clock

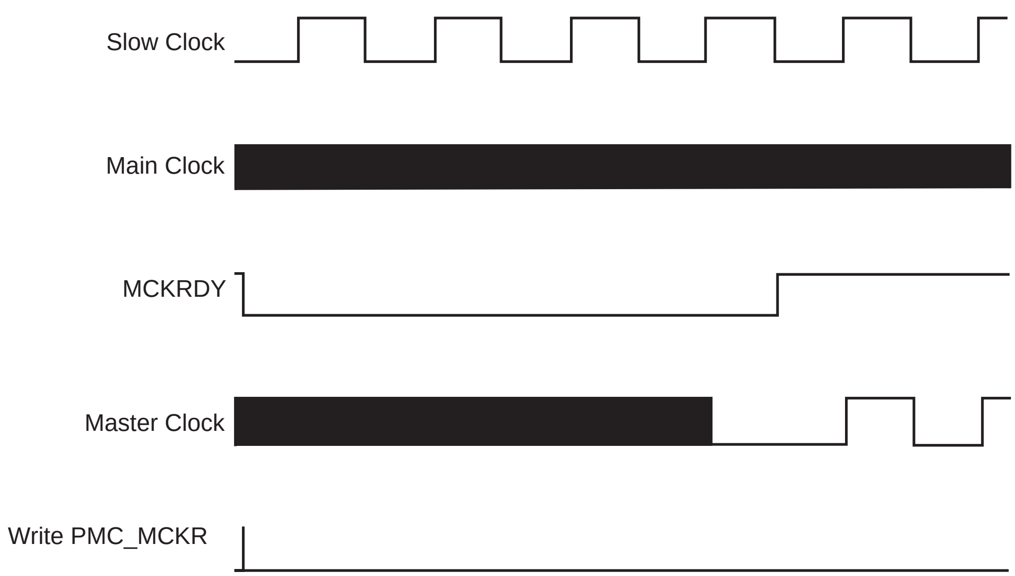


Figure 20-5. Change PLLA Programming

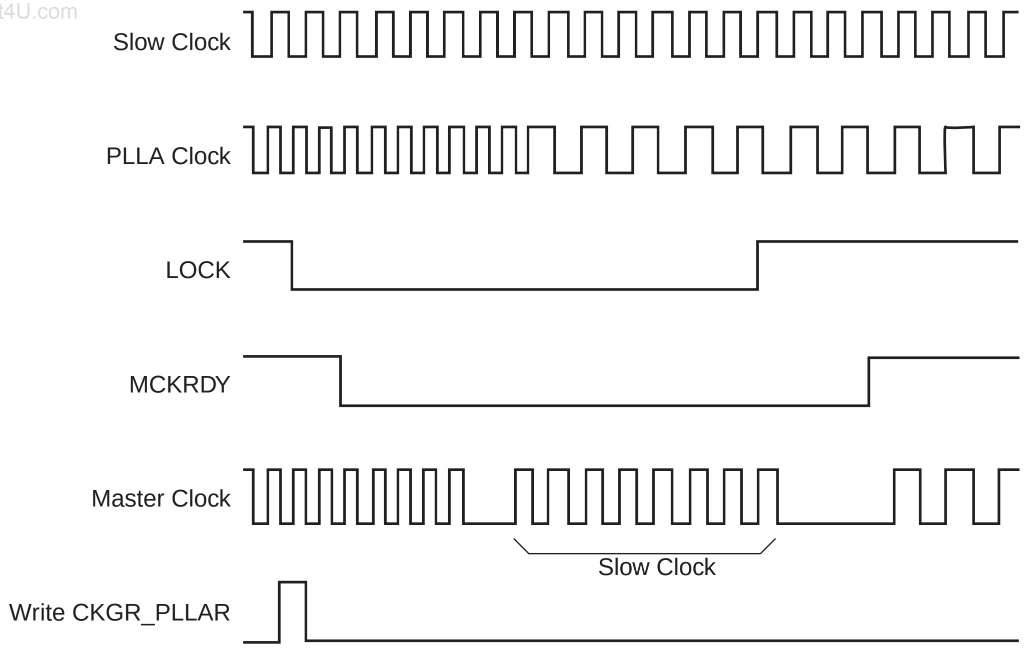


Figure 20-6. Change PLLB Programming

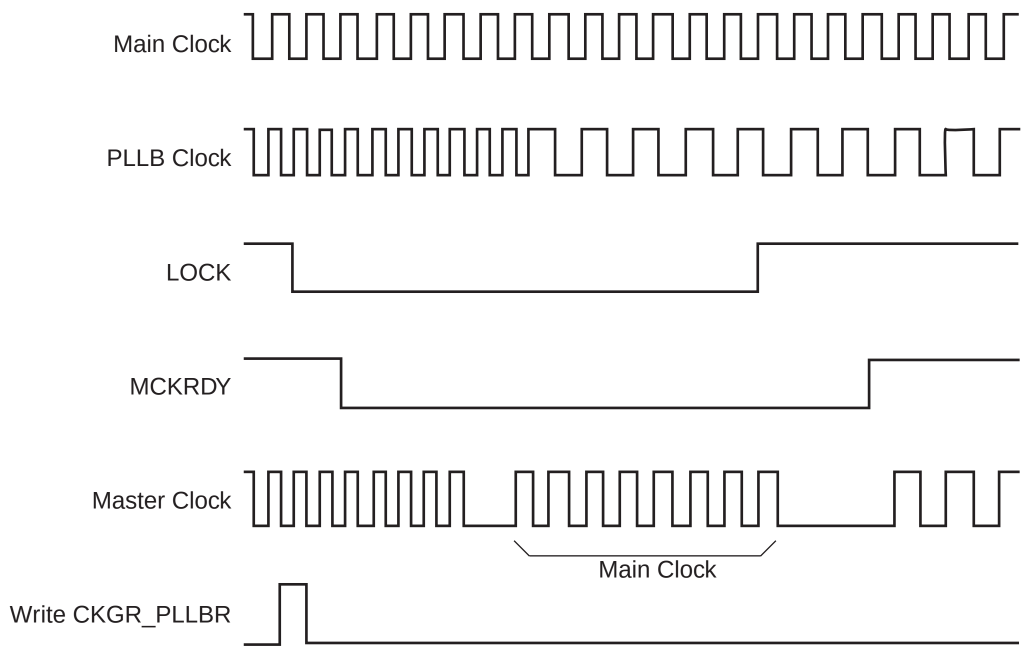
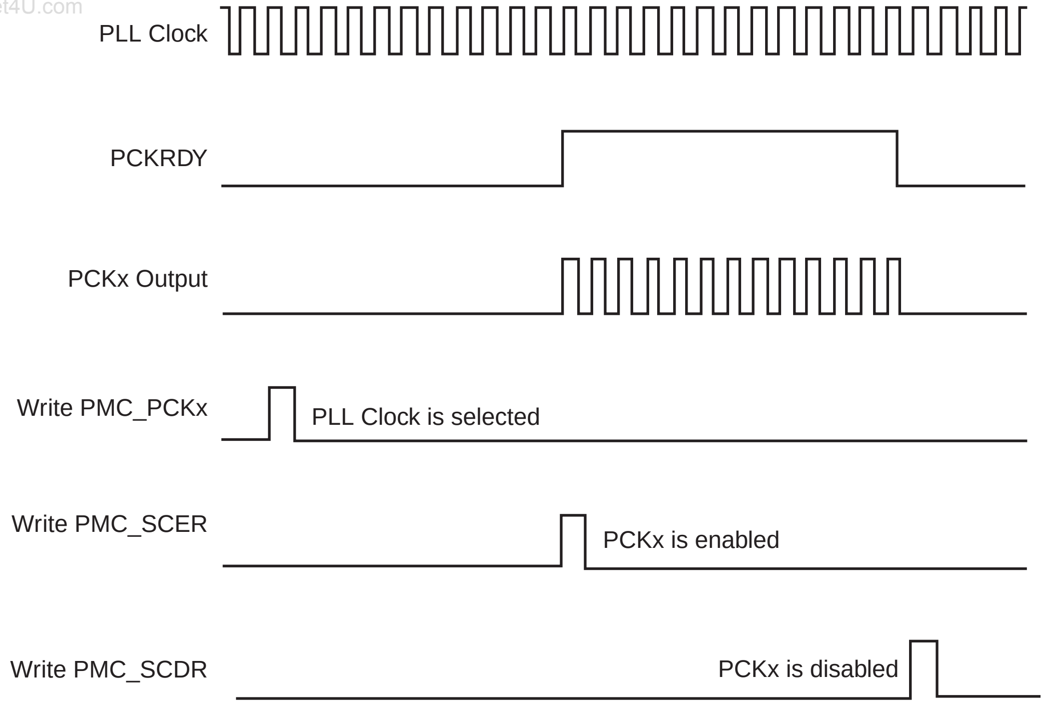


Figure 20-7. Programmable Clock Output Programming



20.9 Power Management Controller (PMC) User Interface

Table 20-3. 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	0x03
0x000C	Reserved	–	–	–
0x0010	Peripheral Clock Enable Register	PMC _PCER	Write-only	–
0x0014	Peripheral Clock Disable Register	PMC_PCDR	Write-only	–
0x0018	Peripheral Clock Status Register	PMC_PCSR	Read-only	0x0
0x001C	Reserved	–	–	–
0x0020	Main Oscillator Register	CKGR_MOR	Read/Write	0x0
0x0024	Main Clock Frequency Register	CKGR_MCFR	Read-only	0x0
0x0028	PLL A Register	CKGR_PLLAR	ReadWrite	0x3F00
0x002C	PLL B Register	CKGR_PLLBR	ReadWrite	0x3F00
0x0030	Master Clock Register	PMC_MCKR	Read/Write	0x0
0x0038	Reserved	–	–	–
0x003C	Reserved	–	–	–
0x0040	Programmable Clock 0 Register	PMC_PCK0	Read/Write	0x0
0x0044	Programmable Clock 1 Register	PMC_PCK1	Read/Write	0x0
...	...	...	...	...
0x0060	Interrupt Enable Register	PMC_IER	Write-only	--
0x0064	Interrupt Disable Register	PMC_IDR	Write-only	--
0x0068	Status Register	PMC_SR	Read-only	0x08
0x006C	Interrupt Mask Register	PMC_IMR	Read-only	0x0
0x0070 - 0x007C	Reserved	–	–	–
0x0080	Charge Pump Current Register	PMC_PLLICPR	Write-only	0x10001
0x0084 - 0x00FC	Reserved	–	–	–

20.9.1 PMC System Clock Enable Register

Register Name: PMC_SCER

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
-	-	-	PCK4	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	UHP	–	–	–	–	-	PCK

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

0 = No effect.

1 = Enables the 12 and 48 MHz clock of the USB Host Port.

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

0 = No effect.

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

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

0 = No effect.

1 = Enables the corresponding Programmable Clock output.

20.9.2 PMC System Clock Disable Register

Register Name: PMC_SCDR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
-	-	-	PCK4	PCK3	PCK2	PCK1	PCK0

7	6	5	4	3	2	1	0
UDP	UHP	–	–	–	–	-	PCK

- **PCK: Processor Clock Disable**

0 = No effect.

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

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

0 = No effect.

1 = Disables the 12 and 48 MHz clock of the USB Host Port.

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

0 = No effect.

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

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

0 = No effect.

1 = Disables the corresponding Programmable Clock output.

20.9.3 PMC System Clock Status Register

Register Name: PMC_SCSR

Access Type: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-
15	14	13	12	11	10	9	8
-	-	-	PCK4	PCK3	PCK2	PCK1	PCK0
7	6	5	4	3	2	1	0
UDP	UHP	–	–	–	–	-	PCK

- **PCK: Processor Clock Status**

0 = The Processor clock is disabled.

1 = The Processor clock is enabled.

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

0 = The 12 and 48 MHz clock (UHPCK) of the USB Host Port is disabled.

1 = The 12 and 48 MHz clock (UHPCK) of the USB Host Port is enabled.

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

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

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

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

0 = The corresponding Programmable Clock output is disabled.

1 = The corresponding Programmable Clock output is enabled.

20.9.4 PMC Peripheral Clock Enable Register

Register Name: PMC_PCER

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PID7	PID6	PID5	PID4	PID3	PID2	-	-

- **PIDx: Peripheral Clock x Enable**

0 = No effect.

1 = Enables the corresponding peripheral clock.

Note: PID2 to PID31 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

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

20.9.5 PMC Peripheral Clock Disable Register

Register Name: PMC_PCDR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PID7	PID6	PID5	PID4	PID3	PID2	-	-

- **PIDx: Peripheral Clock x Disable**

0 = No effect.

1 = Disables the corresponding peripheral clock.

Note: PID2 to PID31 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

20.9.6 PMC Peripheral Clock Status Register

Register Name: PMC_PCSR

Access Type: Read-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	–	–

- **PIDx: Peripheral Clock x Status**

0 = The corresponding peripheral clock is disabled.

1 = The corresponding peripheral clock is enabled.

Note: PID2 to PID31 refer to identifiers as defined in the section “Peripheral Identifiers” in the product datasheet.

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

20.9.7 PMC Clock Generator Main Oscillator Register

Register Name: CKGR_MOR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
OSCOUNT							

7	6	5	4	3	2	1	0
–	–	–	–	–	–	OSCBYPASS	MOSCEN

- **MOSCEN: Main Oscillator Enable**

A crystal must be connected between XIN and XOUT.

0 = The Main Oscillator is disabled.

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

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

- **OSCBYPASS: Oscillator Bypass**

0 = No effect.

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

When OSCBYPASS is set, the MOSCS flag in PMC_SR is automatically set.

Clearing MOSCEN and OSCBYPASS bits allows resetting the MOSCS flag.

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

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

20.9.8 PMC Clock Generator Main Clock Frequency Register

Register Name: CKGR_MCFR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
MAINF							

7	6	5	4	3	2	1	0
MAINF							

- **MAINF: Main Clock Frequency**

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

- **MAINRDY: Main Clock Ready**

0 = MAINF value is not valid or the Main Oscillator is disabled.

1 = The Main Oscillator has been enabled previously and MAINF value is available.

20.9.9 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 PLL A input frequencies and multiplier factors should be checked before using the PMC.

Warning: Bit 29 must always be set to 1 when programming the CKGR_PLLAR register.

- **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: PLL A Clock Frequency Range**

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

- **MULA: PLL A Multiplier**

0 = The PLL A is deactivated.

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

20.9.10 PMC Clock Generator PLL B Register

Register Name: CKGR_PLLBR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
MULB							

15	14	13	12	11	10	9	8
–	–	PLLBCOUNT					

7	6	5	4	3	2	1	0
DIVB							

Possible limitations on PLL B input frequencies and multiplier factors should be checked before using the PMC.

- **DIVB: Divider B**

DIVB	Divider Selected
0	Divider output is 0
1	Divider is bypassed (this value must be selected).
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.

- **MULB: PLL Multiplier**

0 = The PLL B is deactivated.

1 up to 2047 = The PLL B Clock frequency is the PLL B input frequency multiplied by MULB + 1 (MULB must be set to 7)

- **USBDIV: Divider for USB Clock**

USBDIV		Divider for USB Clock(s)
0	0	Divider output is PLL B clock output.
0	1	Divider output is PLL B clock output divided by 2 (this value must be selected).
1	0	Divider output is PLL B clock output divided by 4.
1	1	Reserved.

20.9.11 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	

- **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: Processor Clock Prescaler**

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

- **MDIV: Master Clock Division**

MDIV		Master Clock Division
0	0	Master Clock is Processor Clock.
0	1	Master Clock is Processor Clock divided by 2.
1	0	Master Clock is Processor Clock divided by 4.
1	1	Reserved.

When using MagicV Master Clock Division MDIV must be set to 01 so that MagicV memories clock=ARM processor Clock = 2x AHB system clock = 2x MagicV clock.

20.9.12 PMC Programmable Clock Register

Register Name: PMC_PCKx

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–		PRES			CSS

- **CSS: Master Clock Selection**

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

- **PRES: Programmable Clock Prescaler**

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

PCK4 is MagicV memories clock that must be 2x AHB system clock that is = MagicV clock, so PRES must be the same PRES of Master Clock while Master Clock MDIV must be 1.

20.9.13 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
-	-	-	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
–	-	-	-	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: Main Oscillator Status Interrupt Enable**
- **LOCKA: PLL A Lock Interrupt Enable**
- **LOCKB: PLL B Lock Interrupt Enable**
- **MCKRDY: Master Clock Ready Interrupt Enable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

20.9.14 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
-	-	-	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0

7	6	5	4	3	2	1	0
–	-	-	-	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: Main Oscillator Status Interrupt Disable**
- **LOCKA: PLL A Lock Interrupt Disable**
- **LOCKB: PLL B Lock Interrupt Disable**
- **MCKRDY: Master Clock Ready Interrupt Disable**
- **PCKRDYx: Programmable Clock Ready x Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

20.9.15 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**
-	-	-	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0
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: PLL A Lock Status**

0 = PLL A is not locked

1 = PLL A is locked.

- **LOCKB: PLL B Lock Status**

0 = PLL B is not locked.

1 = PLL B is locked.

- **MCKRDY: Master Clock Status**

0 = Master Clock is not ready.

1 = Master Clock is ready.

- **PCKRDYx: Programmable Clock Ready Status**

0 = Programmable Clock x is not ready.

1 = Programmable Clock x is ready.

20.9.16 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**
-	-	-	PCKRDY4	PCKRDY3	PCKRDY2	PCKRDY1	PCKRDY0
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
–	-	-	-	MCKRDY	LOCKB	LOCKA	MOSCS

- **MOSCS: Main Oscillator Status Interrupt Mask**

- **LOCKA: PLL A Lock Interrupt Mask**
- **LOCKB: PLL B Lock Interrupt Mask**
- **MCKRDY: Master Clock Ready Interrupt Mask**
- **PCKRDYx: Programmable Clock Ready x Interrupt Mask**

0 = The corresponding interrupt is enabled.

1 = The corresponding interrupt is disabled.

20.9.17 PLL Charge Pump Current Register

Register Name: PMC_PLLICPR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	ICPPLLA

- **ICPPLLA: Charge pump current**

Must be set to 1.

- **ICPPLLB: Charge pump current**

Must be set to 1.

21. Advanced Interrupt Controller (AIC)

21.1 Description

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.

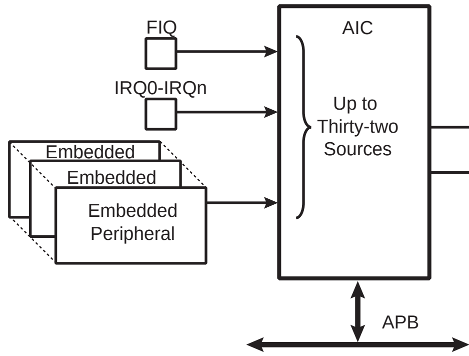
The 8-level Priority Controller allows the user to define the priority for each interrupt source, thus permitting higher priority interrupts to be serviced even if a lower priority interrupt is being treated.

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.

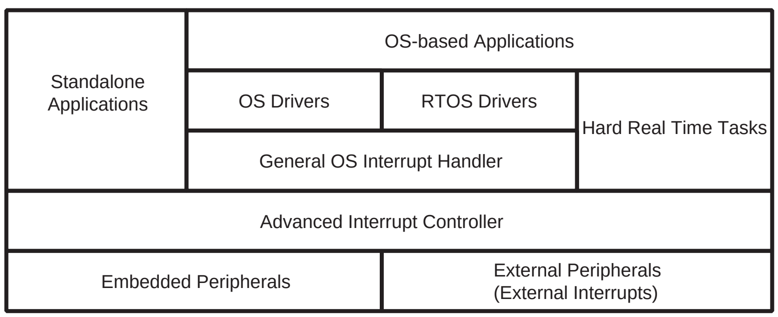
21.2 Block Diagram

Figure 21-1. Block Diagram



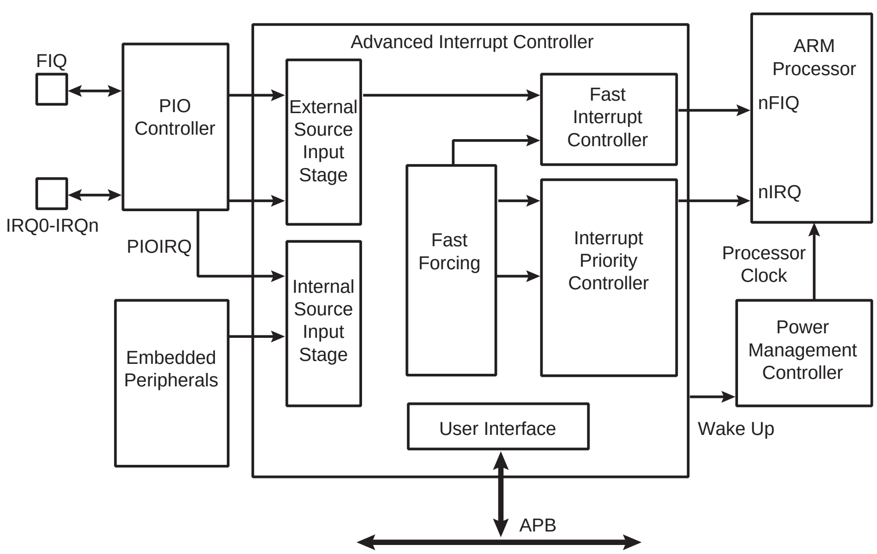
21.3 Application Block Diagram

Figure 21-2. Description of the Application Block



21.4 AIC Detailed Block Diagram

Figure 21-3. AIC Detailed Block Diagram



21.5 I/O Line Description

Table 21-1. I/O Line Description

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

21.6 Product Dependencies

21.6.1 I/O Lines

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

21.6.2 Power Management

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

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

21.6.3 Interrupt Sources

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

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

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

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

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

21.7 Functional Description

21.7.1 Interrupt Source Control

21.7.1.1 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.

21.7.1.2 Interrupt Source Enabling

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

21.7.1.3 Interrupt Clearing and Setting

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

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

The AIC features an automatic clear of the current interrupt when the AIC_IVR (Interrupt Vector Register) is read. Only the interrupt source being detected by the AIC as the current interrupt is affected by this operation. (See "Priority Controller" on page 271.) 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 275.)

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

21.7.1.4 Interrupt Status

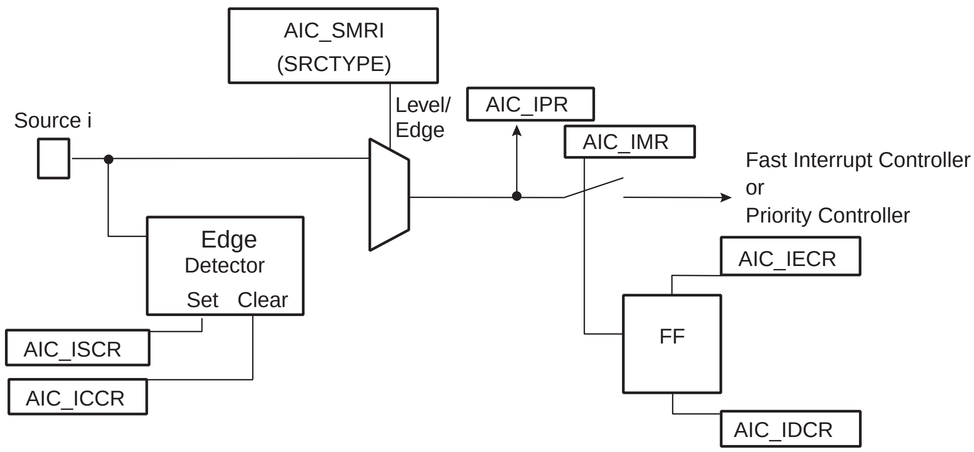
For each interrupt, the AIC operation originates in AIC_IPR (Interrupt Pending Register) and its mask in AIC_IMR (Interrupt Mask Register). AIC_IPR enables the actual activity of the sources, whether masked or not.

The AIC_ISR register reads the number of the current interrupt (see "Priority Controller" on page 271) 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.

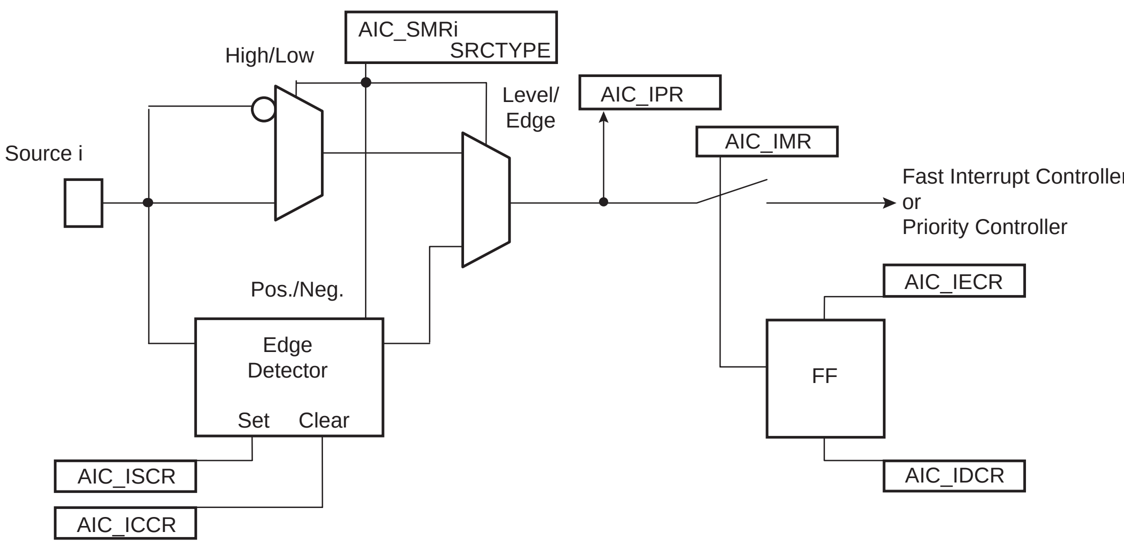
21.7.1.5 Internal Interrupt Source Input Stage

Figure 21-4. Internal Interrupt Source Input Stage



21.7.1.6 External Interrupt Source Input Stage

Figure 21-5. External Interrupt Source Input Stage



21.7.2 Interrupt Latencies

Global interrupt latencies depend on several parameters, including:

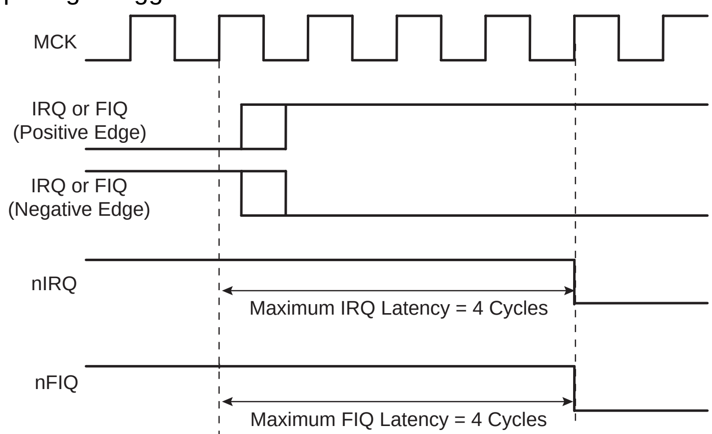
- The time the software masks the interrupts.
- Occurrence, either at the processor level or at the AIC level.
- The execution time of the instruction in progress when the interrupt occurs.
- The treatment of higher priority interrupts and the resynchronization of the hardware signals.

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

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

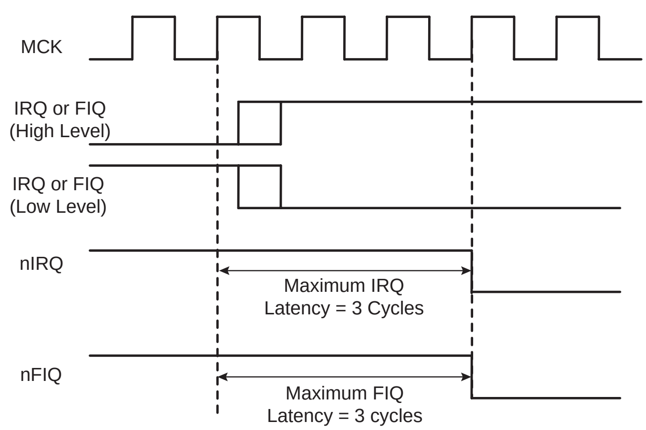
21.7.2.1 External Interrupt Edge Triggered Source

Figure 21-6. External Interrupt Edge Triggered Source



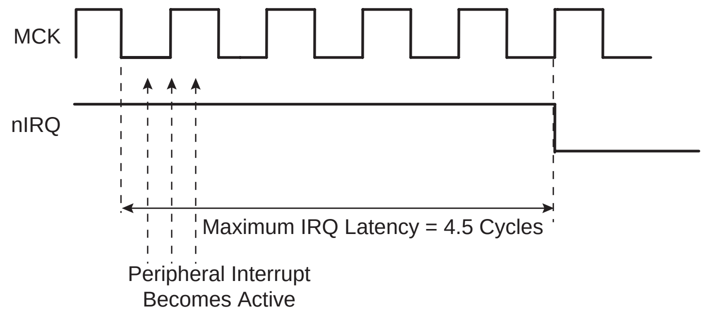
21.7.2.2 External Interrupt Level Sensitive Source

Figure 21-7. External Interrupt Level Sensitive Source



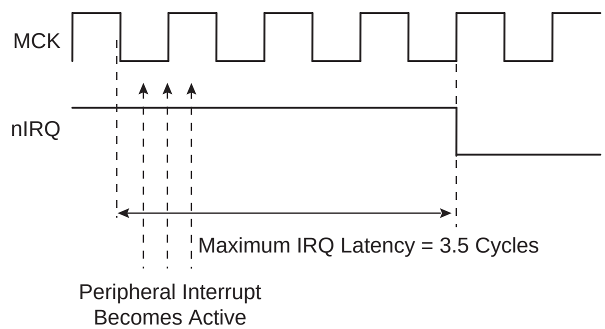
21.7.2.3 Internal Interrupt Edge Triggered Source

Figure 21-8. Internal Interrupt Edge Triggered Source



21.7.2.4 Internal Interrupt Level Sensitive Source

Figure 21-9. Internal Interrupt Level Sensitive Source



21.7.3 Normal Interrupt

21.7.3.1 Priority Controller

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

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

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

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

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

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

21.7.3.2 *Interrupt Nesting*

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

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

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

21.7.3.3 *Interrupt Vectoring*

The interrupt handler addresses corresponding to each interrupt source can be stored in the registers AIC_SVR1 to AIC_SVR31 (Source Vector Register 1 to 31). When the processor reads AIC_IVR (Interrupt Vector Register), the value written into AIC_SVR corresponding to the current interrupt is returned.

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

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

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

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

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

21.7.3.4 *Interrupt Handlers*

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

It is assumed that:

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

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

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

1. The CPSR is stored in SPSR_irq, the current value of the Program Counter is loaded in the Interrupt link register (R14_irq) and the Program Counter (R15) is loaded with 0x18. In the following cycle during fetch at address 0x1C, the ARM core adjusts R14_irq, decrementing it by four.
2. The ARM core enters Interrupt mode, if it has not already done so.
3. When the instruction loaded at address 0x18 is executed, the program counter is loaded with the value read in AIC_IVR. Reading the AIC_IVR has the following effects:
   – Sets the current interrupt to be the pending and enabled interrupt with the highest priority. The current level is the priority level of the current interrupt.
   – De-asserts the nIRQ line on the processor. Even if vectoring is not used, AIC_IVR must be read in order to de-assert nIRQ.
   – Automatically clears the interrupt, if it has been programmed to be edge-triggered.
   – Pushes the current level and the current interrupt number on to the stack.
   – Returns the value written in the AIC_SVR corresponding to the current interrupt.
4. The previous step has the effect of branching to the corresponding interrupt service routine. This should start by saving the link register (R14_irq) and SPSR_IRQ. The link register must be decremented by four when it is saved if it is to be restored directly into the program counter at the end of the interrupt. For example, the instruction `SUB PC, LR, #4` may be used.
5. Further interrupts can then be unmasked by clearing the “I” bit in CPSR, allowing re-assertion of the nIRQ to be taken into account by the core. This can happen if an interrupt with a higher priority than the current interrupt occurs.
6. The interrupt handler can then proceed as required, saving the registers that will be used and restoring them at the end. During this phase, an interrupt of higher priority than the current level will restart the sequence from step 1.

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

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

executed before, and of loading the CPSR with the stored SPSR, masking or unmasking the interrupts depending on the state saved in SPSR_irq.

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

21.7.4 Fast Interrupt

21.7.4.1 Fast Interrupt Source

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

21.7.4.2 Fast Interrupt Control

The fast interrupt logic of the AIC has no priority controller. The mode of interrupt source 0 is programmed with the AIC_SMR0 and the field PRIOR of this register is not used even if it reads what has been written. The field SRCTYPE of AIC_SMR0 enables programming the fast interrupt source to be positive-edge triggered or negative-edge triggered or high-level sensitive or low-level sensitive

Writing 0x1 in the AIC_IECR (Interrupt Enable Command Register) and AIC_IDCR (Interrupt Disable Command Register) respectively enables and disables the fast interrupt. The bit 0 of AIC_IMR (Interrupt Mask Register) indicates whether the fast interrupt is enabled or disabled.

21.7.4.3 Fast Interrupt Vectoring

The fast interrupt handler address can be stored in AIC_SVR0 (Source Vector Register 0). The value written into this register is returned when the processor reads AIC_FVR (Fast Vector Register). This offers a way to branch in one single instruction to the interrupt handler, as AIC_FVR is mapped at the absolute address 0xFFFF F104 and thus accessible from the ARM fast interrupt vector at address 0x0000 001C through the following instruction:

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

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

21.7.4.4 Fast Interrupt Handlers

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

Assuming that:

1. The Advanced Interrupt Controller has been programmed, AIC_SVR0 is loaded with the fast interrupt service routine address, and the interrupt source 0 is enabled.
2. The Instruction at address 0x1C (FIQ exception vector address) is required to vector the fast interrupt:

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

3. The user does not need nested fast interrupts.

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

1. The CPSR is stored in SPSR_fiq, the current value of the program counter is loaded in the FIQ link register (R14_FIQ) and the program counter (R15) is loaded with 0x1C. In the following cycle, during fetch at address 0x20, the ARM core adjusts R14_fiq, decrementing it by four.
2. The ARM core enters FIQ mode.
3. When the instruction loaded at address 0x1C is executed, the program counter is loaded with the value read in AIC_FVR. Reading the AIC_FVR has effect of 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.

21.7.4.5 Fast Forcing

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

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

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

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

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

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

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

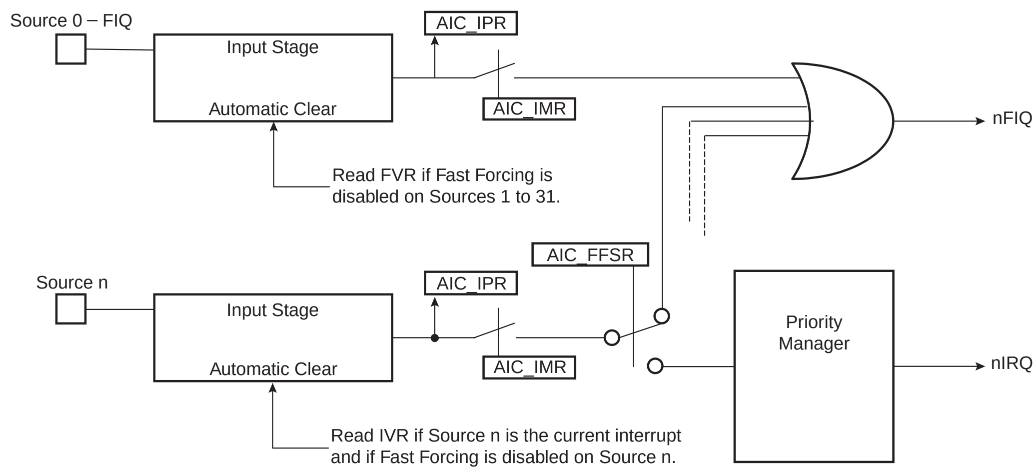
The FIQ Vector Register (AIC_FVR) reads the contents of the Source Vector Register 0 (AIC_SVR0), whatever the source of the fast interrupt may be. The read of the FVR does not clear the Source 0 when the fast forcing feature is used and the interrupt source should be cleared by writing to the Interrupt Clear Command Register (AIC_ICCR).

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

The read of AIC_IVR does not clear the source that has the fast forcing feature enabled.

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

Figure 21-10. Fast Forcing



21.7.5 Protect Mode

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

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

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

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

When the Protect Mode is enabled, the AIC performs interrupt stacking only when a write access is performed on the AIC_IVR. Therefore, the Interrupt Service Routines must write (arbitrary data) to the AIC_IVR just after reading it. The new context of the AIC, including the value of the

Interrupt Status Register (AIC_ISR), is updated with the current interrupt only when AIC_IVR is written.

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

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

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

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

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

21.7.6 Spurious Interrupt

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

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

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

21.7.7 General Interrupt Mask

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

21.8 Advanced Interrupt Controller (AIC) User Interface

21.8.1 Base Address

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

21.8.2 Register Mapping

Table 21-2. Register Mapping

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

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

21.8.3 AIC Source Mode Register

Register Name: AIC_SMR0..AIC_SMR31

Access Type: Read/Write

Reset Value: 0x0

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	SRCTYPE		–	–	PRIOR		

- **PRIOR: Priority Level**

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

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

The priority level is not used for the FIQ in the related SMR register AIC_SMRx.

- **SRCTYPE: Interrupt Source Type**

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

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

21.8.4 AIC Source Vector Register

Register Name: AIC_SVR0..AIC_SVR31

Access Type: Read/Write

Reset Value: 0x0

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

- **VECTOR: Source Vector**

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

21.8.5 AIC Interrupt Vector Register

Register Name: AIC_IVR

Access Type: Read-only

Reset Value: 0x0

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

- **IRQV: Interrupt Vector Register**

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

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

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

21.8.6 AIC FIQ Vector Register

Register Name: AIC_FVR

Access Type: Read-only

Reset Value: 0x0

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

- **FIQV: FIQ Vector Register**

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

21.8.7 AIC Interrupt Status Register

Register Name: AIC_ISR

Access Type: Read-only

Reset Value: 0x0

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

- **IRQID: Current Interrupt Identifier**

The Interrupt Status Register returns the current interrupt source number.

21.8.8 AIC Interrupt Pending Register

Register Name: AIC_IPR

Access Type: Read-only

Reset Value: 0x0

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

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

0 = Corresponding interrupt is not pending.

1 = Corresponding interrupt is pending.

21.8.9 AIC Interrupt Mask Register

Register Name: AIC_IMR

Access Type: Read-only

Reset Value: 0x0

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

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

0 = Corresponding interrupt is disabled.

1 = Corresponding interrupt is enabled.

21.8.10 AIC Core Interrupt Status Register

Register Name: AIC_CISR

Access Type: Read-only

Reset Value: 0x0

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

21.8.11 AIC Interrupt Enable Command Register

Register Name: AIC_IECR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

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

0 = No effect.

1 = Enables corresponding interrupt.

21.8.12 AIC Interrupt Disable Command Register

Register Name: AIC_IDCR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24

23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16

15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8

7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

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

0 = No effect.

1 = Disables corresponding interrupt.

21.8.13 AIC Interrupt Clear Command Register

Register Name: AIC_ICCR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24

23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16

15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8

7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

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

0 = No effect.

1 = Clears corresponding interrupt.

21.8.14 AIC Interrupt Set Command Register

Register Name: AIC_ISCR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PID7	PID6	PID5	PID4	PID3	PID2	SYS	FIQ

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

0 = No effect.

1 = Sets corresponding interrupt.

21.8.15 AIC End of Interrupt Command Register

Register Name: AIC_EOICR

Access Type: Write-only

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

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

21.8.16 AIC Spurious Interrupt Vector Register

Register Name: AIC_SPU

Access Type: Read/Write

Reset Value: 0x0

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

- **SIQV: Spurious Interrupt Vector Register**

The user may store the address of a spurious interrupt handler in this register. The written value is returned in AIC_IVR in case of a spurious interrupt and in AIC_FVR in case of a spurious fast interrupt.

21.8.17 AIC Debug Control Register

Register Name: AIC_DEBUG

Access Type: Read/Write

Reset Value: 0x0

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

- **PROT: Protection Mode**

0 = The Protection Mode is disabled.

1 = The Protection Mode is enabled.

- **GMSK: General Mask**

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

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

21.8.18 AIC Fast Forcing Enable Register

Register Name: AIC_FFER

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24

23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16

15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8

7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	–

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

0 = No effect.

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

21.8.19 AIC Fast Forcing Disable Register

Register Name: AIC_FFDR

Access Type: Write-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24

23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16

15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8

7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	–

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

0 = No effect.

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

21.8.20 AIC Fast Forcing Status Register

Register Name: AIC_FFSR

Access Type: Read-only

31	30	29	28	27	26	25	24
PID31	PID30	PID29	PID28	PID27	PID26	PID25	PID24
23	22	21	20	19	18	17	16
PID23	PID22	PID21	PID20	PID19	PID18	PID17	PID16
15	14	13	12	11	10	9	8
PID15	PID14	PID13	PID12	PID11	PID10	PID9	PID8
7	6	5	4	3	2	1	0
PID7	PID6	PID5	PID4	PID3	PID2	SYS	–

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

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

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

22. Debug Unit (DBGU)

22.1 Description

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 A_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.

22.2 Block Diagram

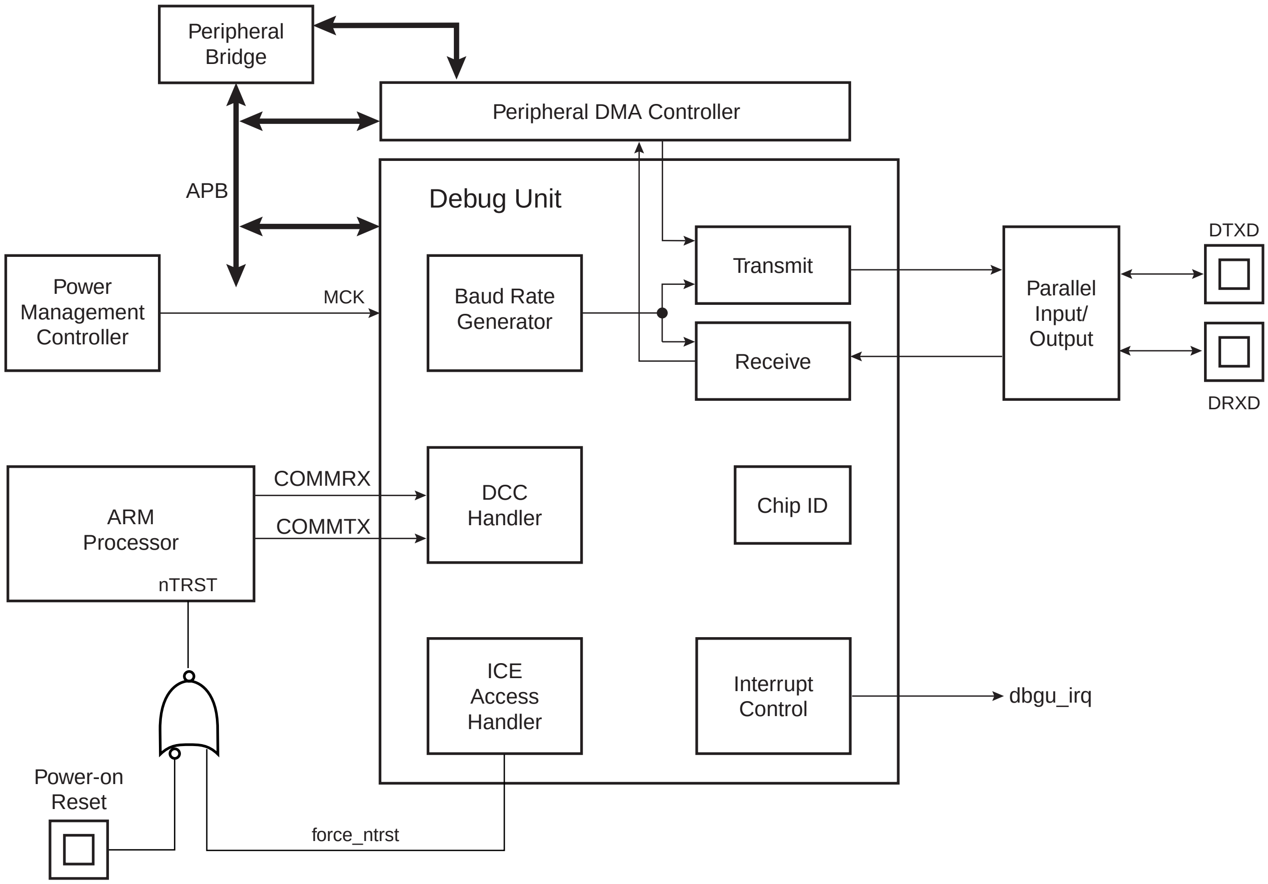
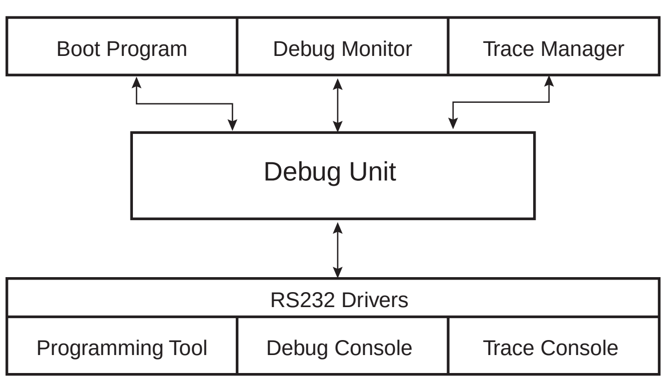


Figure 22-1. Debug Unit Functional Block Diagram

Table 22-1. Debug Unit Pin Description

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

Figure 22-2. Debug Unit Application Example



22.3 Product Dependencies

22.3.1 I/O Lines

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

22.3.2 Power Management

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

22.3.3 Interrupt Source

Depending on product integration, the Debug Unit interrupt line is connected to one of the interrupt sources of the Advanced Interrupt Controller. Interrupt handling requires programming of the AIC before configuring the Debug Unit. Usually, the Debug Unit interrupt line connects to the interrupt source 1 of the AIC, which may be shared with the real-time clock, the system timer interrupt lines and other system peripheral interrupts, as shown in Figure 22-1. This sharing requires the programmer to determine the source of the interrupt when the source 1 is triggered.

22.4 UART Operations

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

The Debug Unit's UART is made up of a receiver and a transmitter that operate independently, and a common baud rate generator. Receiver timeout and transmitter time guard are not implemented. However, all the implemented features are compatible with those of a standard USART.

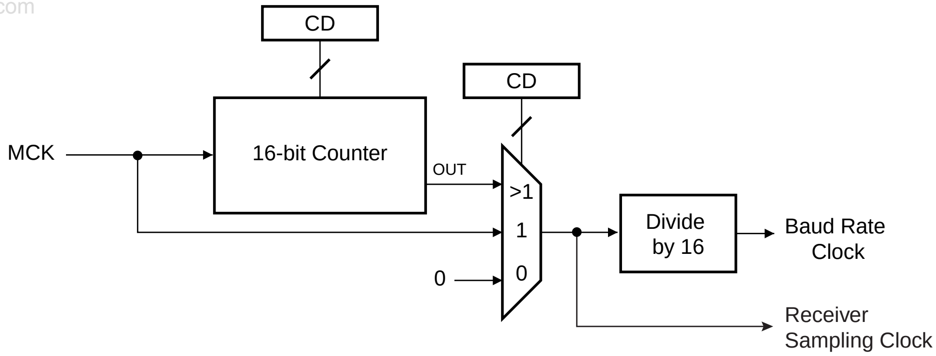
22.4.1 Baud Rate Generator

The baud rate generator provides the bit period clock named baud rate clock to both the receiver and the transmitter.

The baud rate clock is the master clock divided by 16 times the value (CD) written in DBGU_BRGR (Baud Rate Generator Register). If DBGU_BRGR is set to 0, the baud rate clock is disabled and the Debug Unit's UART remains inactive. The maximum allowable baud rate is Master Clock divided by 16. The minimum allowable baud rate is Master Clock divided by (16 x 65536).

$$\text{Baud Rate} = \frac{MCK}{16 \times CD}$$

Figure 22-3. Baud Rate Generator



22.4.2 Receiver

22.4.2.1 Receiver Reset, Enable and Disable

After device reset, the Debug Unit receiver is disabled and must be enabled before being used. The receiver can be enabled by writing the control register DBGU_CR with the bit RXEN at 1. At this command, the receiver starts looking for a start bit.

The programmer can disable the receiver by writing DBGU_CR with the bit RXDIS at 1. If the receiver is waiting for a start bit, it is immediately stopped. However, if the receiver has already detected a start bit and is receiving the data, it waits for the stop bit before actually stopping its operation.

The programmer can also put the receiver in its reset state by writing DBGU_CR with the bit RSTRX at 1. In doing so, the receiver immediately stops its current operations and is disabled, whatever its current state. If RSTRX is applied when data is being processed, this data is lost.

22.4.2.2 Start Detection and Data Sampling

The Debug Unit only supports asynchronous operations, and this affects only its receiver. The Debug Unit receiver detects the start of a received character by sampling the DRXD signal until it detects a valid start bit. A low level (space) on DRXD is interpreted as a valid start bit if it is detected for more than 7 cycles of the sampling clock, which is 16 times the baud rate. Hence, a space that is longer than 7/16 of the bit period is detected as a valid start bit. A space which is 7/16 of a bit period or shorter is ignored and the receiver continues to wait for a valid start bit.

When a valid start bit has been detected, the receiver samples the DRXD at the theoretical mid-point 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 22-4. Start Bit Detection

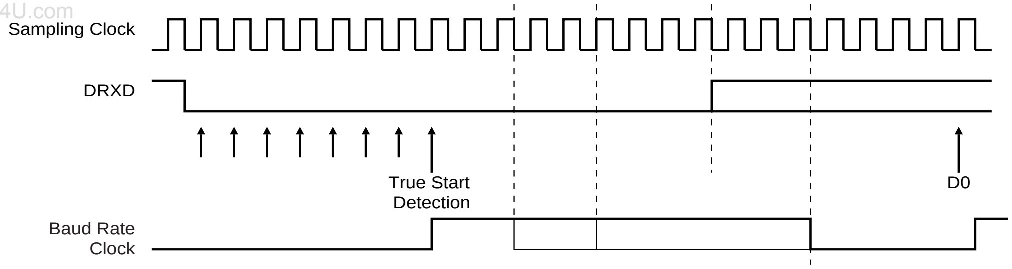
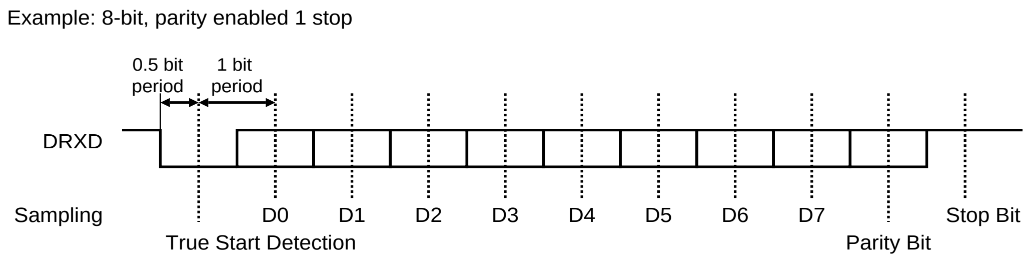


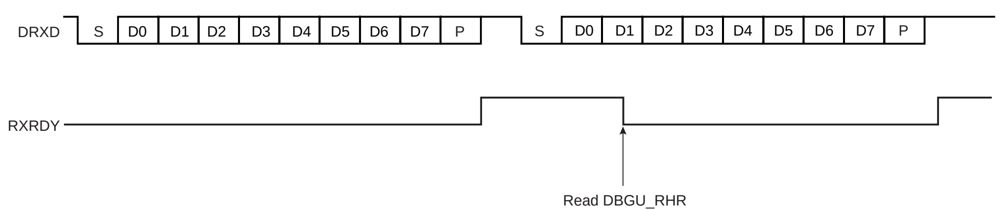
Figure 22-5. Character Reception



22.4.2.3 Receiver Ready

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

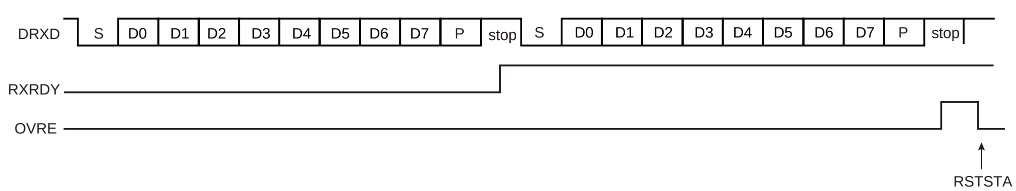
Figure 22-6. Receiver Ready



22.4.2.4 Receiver Overrun

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

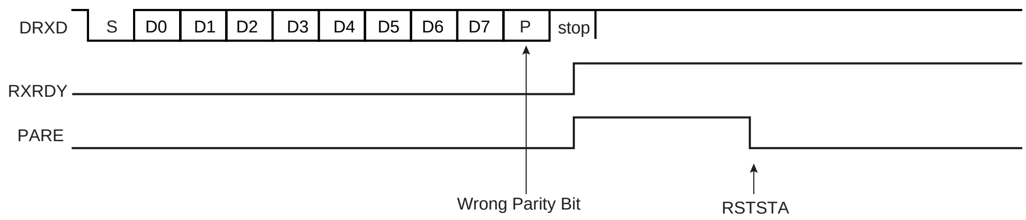
Figure 22-7. Receiver Overrun



22.4.2.5 Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits, in accordance with the field PAR in DBGU_MR. It then compares the result with the received parity bit. If different, the parity error bit PARE in DBGU_SR is set at the same time the RXRDY is set. The parity bit is cleared when the control register DBGU_CR is written with the bit RSTSTA (Reset Status) at 1. If a new character is received before the reset status command is written, the PARE bit remains at 1.

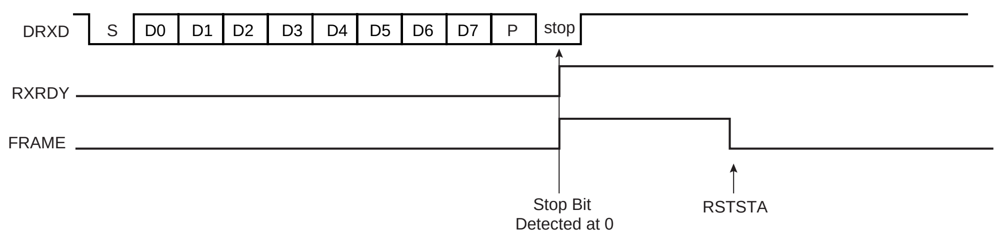
Figure 22-8. Parity Error



22.4.2.6 Receiver Framing Error

When a start bit is detected, it generates a character reception when all the data bits have been sampled. The stop bit is also sampled and when it is detected at 0, the FRAME (Framing Error) bit in DBGU_SR is set at the same time the RXRDY bit is set. The bit FRAME remains high until the control register DBGU_CR is written with the bit RSTSTA at 1.

Figure 22-9. Receiver Framing Error



22.4.3 Transmitter

22.4.3.1 Transmitter Reset, Enable and Disable

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

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

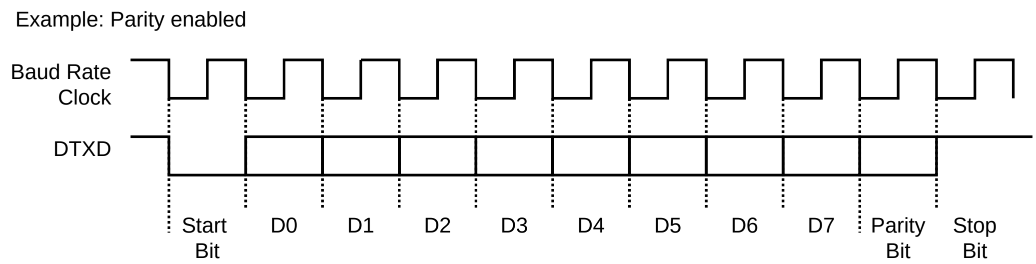
The programmer can also put the transmitter in its reset state by writing the DBGU_CR with the bit RSTTX at 1. This immediately stops the transmitter, whether or not it is processing characters.

22.4.3.2 Transmit Format

The Debug Unit transmitter drives the pin DTXD at the baud rate clock speed. The line is driven depending on the format defined in the Mode Register and the data stored in the Shift Register. One start bit at level 0, then the 8 data bits, from the lowest to the highest bit, one optional parity bit and one stop bit at 1 are consecutively shifted out as shown on the following figure. The field

PARE in the mode register DBGU_MR defines whether or not a parity bit is shifted out. When a parity bit is enabled, it can be selected between an odd parity, an even parity, or a fixed space or mark bit.

Figure 22-10. Character Transmission

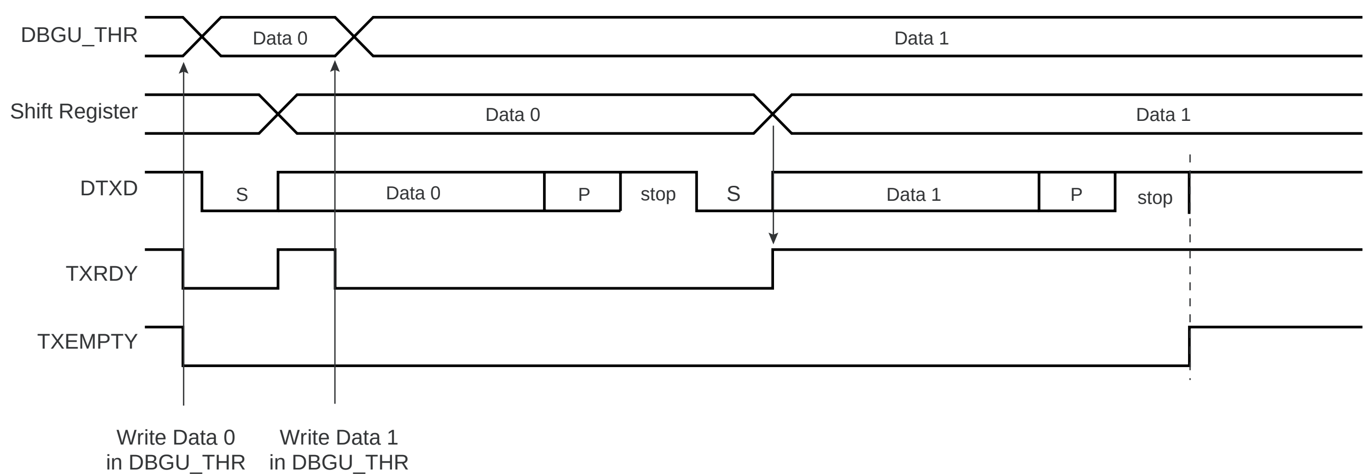


22.4.3.3 Transmitter Control

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

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

Figure 22-11. Transmitter Control



22.4.4 Peripheral Data Controller

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

The peripheral data controller channels are programmed via registers that are mapped within the Debug Unit user interface from the offset 0x100. The status bits are reported in the Debug Unit status register DBGU_SR and can generate an interrupt.

The RXRDY bit triggers the PDC channel data transfer of the receiver. This results in a read of the data in DBGU_RHR. The TXRDY bit triggers the PDC channel data transfer of the transmitter. This results in a write of a data in DBGU_THR.

22.4.5 Test Modes

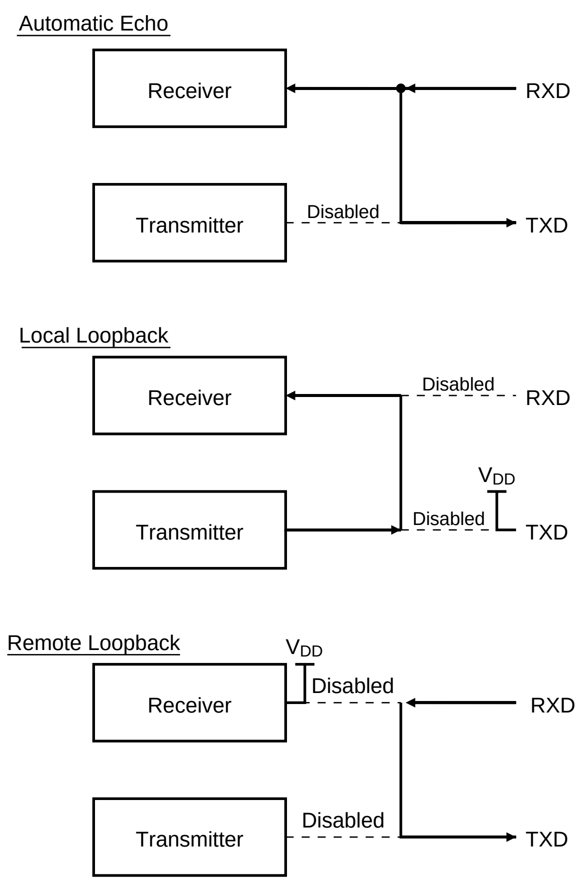
The Debug Unit supports three tests modes. These modes of operation are programmed by using the field CHMODE (Channel Mode) in the mode register DBGU_MR.

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

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

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

Figure 22-12. Test Modes



22.4.6 Debug Communication Channel Support

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

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

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

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

Returns the debug communication data read register into Rd

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

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

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

22.4.7 Chip Identifier

The Debug Unit features two chip identifier registers, DBGU_CIDR (Chip ID Register) and DBGU_EXID (Extension ID). Both registers contain a hard-wired value that is read-only. The first register contains the following fields:

- EXT - shows the use of the extension identifier register
- NVPTYP and NVPSIZ - identifies the type of embedded non-volatile memory and its size
- ARCH - identifies the set of embedded peripherals
- SRAMSIZ - indicates the size of the embedded SRAM
- EPROC - indicates the embedded ARM processor
- VERSION - gives the revision of the silicon

The second register is device-dependent and reads 0 if the bit EXT is 0.

22.4.8 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 A_NTRST (DBGU_FNR), that allows assertion of the A_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.

22.5 Debug Unit User Interface

Table 22-2. Debug Unit Memory Map

Offset	Register	Name	Access	Reset Value
0x0000	Control Register	DBGU_CR	Write-only	–
0x0004	Mode Register	DBGU_MR	Read/Write	0x0
0x0008	Interrupt Enable Register	DBGU_IER	Write-only	–
0x000C	Interrupt Disable Register	DBGU_IDR	Write-only	–
0x0010	Interrupt Mask Register	DBGU_IMR	Read-only	0x0
0x0014	Status Register	DBGU_SR	Read-only	–
0x0018	Receive Holding Register	DBGU_RHR	Read-only	0x0
0x001C	Transmit Holding Register	DBGU_THR	Write-only	–
0x0020	Baud Rate Generator Register	DBGU_BRGR	Read/Write	0x0
0x0024 - 0x003C	Reserved	–	–	–
0x0040	Chip ID Register	DBGU_CIDR	Read-only	0x0E0303E0 (1)
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	–	–	–

1. CIDR bit 0 reset value is 0 for D940HF rev A, 1 for D940HF rev B.

22.5.1 Debug Unit Control Register

Name: DBGU_CR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	–	–	–	–
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	–	–	–	–	RSTSTA
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **RSTRX: Reset Receiver**

0 = No effect.

1 = The receiver logic is reset and disabled. If a character is being received, the reception is aborted.

- **RSTTX: Reset Transmitter**

0 = No effect.

1 = The transmitter logic is reset and disabled. If a character is being transmitted, the transmission is aborted.

- **RXEN: Receiver Enable**

0 = No effect.

1 = The receiver is enabled if RXDIS is 0.

- **RXDIS: Receiver Disable**

0 = No effect.

1 = The receiver is disabled. If a character is being processed and RSTRX is not set, the character is completed before the receiver is stopped.

- **TXEN: Transmitter Enable**

0 = No effect.

1 = The transmitter is enabled if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0 = No effect.

1 = The transmitter is disabled. If a character is being processed and a character has been written the DBGU_THR and RSTTX is not set, both characters are completed before the transmitter is stopped.

- **RSTSTA: Reset Status Bits**

0 = No effect.

1 = Resets the status bits PARE, FRAME and OVRE in the DBGU_SR.

22.5.2 Debug Unit Mode Register

Name: DBGU_MR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
CHMODE		–	–	PAR			–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **PAR: Parity Type**

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

- **CHMODE: Channel Mode**

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

22.5.3 Debug Unit Interrupt Enable Register

Name: DBGU_IER

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–

7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Enable RXRDY Interrupt**
- **TXRDY: Enable TXRDY Interrupt**
- **ENDRX: Enable End of Receive Transfer Interrupt**
- **ENDTX: Enable End of Transmit Interrupt**
- **OVRE: Enable Overrun Error Interrupt**
- **FRAME: Enable Framing Error Interrupt**
- **PARE: Enable Parity Error Interrupt**
- **TXEMPTY: Enable TXEMPTY Interrupt**
- **TXBUFE: Enable Buffer Empty Interrupt**
- **RXBUFF: Enable Buffer Full Interrupt**
- **COMMTX: Enable COMMTX (from ARM) Interrupt**
- **COMMRX: Enable COMMRX (from ARM) Interrupt**

0 = No effect.

1 = Enables the corresponding interrupt.

22.5.4 Debug Unit Interrupt Disable Register

Name: DBGU_IDR

Access Type: Write-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	–	–	–	–
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Disable RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **ENDRX: Disable End of Receive Transfer Interrupt**
- **ENDTX: Disable End of Transmit Interrupt**
- **OVRE: Disable Overrun Error Interrupt**
- **FRAME: Disable Framing Error Interrupt**
- **PARE: Disable Parity Error Interrupt**
- **TXEMPTY: Disable TXEMPTY Interrupt**
- **TXBUFE: Disable Buffer Empty Interrupt**
- **RXBUFF: Disable Buffer Full Interrupt**
- **COMMTX: Disable COMMTX (from ARM) Interrupt**
- **COMMRX: Disable COMMRX (from ARM) Interrupt**

0 = No effect.

1 = Disables the corresponding interrupt.

22.5.5 Debug Unit Interrupt Mask Register

Name: DBGU_IMR

Access Type: Read-only

31	30	29	28	27	26	25	24
COMMRX	COMMTX	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	–	–	–	–
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Mask RXRDY Interrupt**
- **TXRDY: Disable TXRDY Interrupt**
- **ENDRX: Mask End of Receive Transfer Interrupt**
- **ENDTX: Mask End of Transmit Interrupt**
- **OVRE: Mask Overrun Error Interrupt**
- **FRAME: Mask Framing Error Interrupt**
- **PARE: Mask Parity Error Interrupt**
- **TXEMPTY: Mask TXEMPTY Interrupt**
- **TXBUFE: Mask TXBUFE Interrupt**
- **RXBUFF: Mask RXBUFF Interrupt**
- **COMMTX: Mask COMMTX Interrupt**
- **COMMRX: Mask COMMRX Interrupt**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

22.5.6 Debug Unit Status Register

Name: DBGU_SR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	RXBUFF	TXBUFE	–	TXEMPTY	–

7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	–	TXRDY	RXRDY

- **RXRDY: Receiver Ready**

0 = No character has been received since the last read of the DBGU_RHR or the receiver is disabled.

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

- **TXRDY: Transmitter Ready**

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

1 = There is no character written to DBGU_THR not yet transferred to the Shift Register.

- **ENDRX: End of Receiver Transfer**

0 = The End of Transfer signal from the receiver Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the receiver Peripheral Data Controller channel is active.

- **ENDTX: End of Transmitter Transfer**

0 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is inactive.

1 = The End of Transfer signal from the transmitter Peripheral Data Controller channel is active.

- **OVRE: Overrun Error**

0 = No overrun error has occurred since the last RSTSTA.

1 = At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0 = No framing error has occurred since the last RSTSTA.

1 = At least one framing error has occurred since the last RSTSTA.

- **PARE: Parity Error**

0 = No parity error has occurred since the last RSTSTA.

1 = At least one parity error has occurred since the last RSTSTA.

- **TXEMPTY: Transmitter Empty**

0 = There are characters in DBGU_THR, or characters being processed by the transmitter, or the transmitter is disabled.

1 = There are no characters in DBGU_THR and there are no characters being processed by the transmitter.

- **TXBUFE: Transmission Buffer Empty**

0 = The buffer empty signal from the transmitter PDC channel is inactive.

1 = The buffer empty signal from the transmitter PDC channel is active.

- **RXBUFF: Receive Buffer Full**

0 = The buffer full signal from the receiver PDC channel is inactive.

1 = The buffer full signal from the receiver PDC channel is active.

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

0 = COMMTX from the ARM processor is inactive.

1 = COMMTX from the ARM processor is active.

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

0 = COMMRX from the ARM processor is inactive.

1 = COMMRX from the ARM processor is active.

22.5.7 Debug Unit Receiver Holding Register

Name: DBGU_RHR

Access Type: Read-only

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

- **RXCHR: Received Character**

Last received character if RXRDY is set.

22.5.8 Debug Unit Transmit Holding Register

Name: DBGU_THR

Access Type: Write-only

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

- **TXCHR: Character to be Transmitted**

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

22.5.9 Debug Unit Baud Rate Generator Register

Name: DBGU_BRGR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
CD							

7	6	5	4	3	2	1	0
CD							

- **CD: Clock Divisor**

CD	Baud Rate Clock
0	Disabled
1	MCK
2 to 65535	MCK / (CD x 16)

22.5.10 Debug Unit Chip ID Register

Name: DBGU_CIDR

Access Type: Read-only

31	30	29	28	27	26	25	24
EXT	NVPTYP			ARCH			

23	22	21	20	19	18	17	16
ARCH				SRAMSIZ			

15	14	13	12	11	10	9	8
NVPSIZ2				NVPSIZ			

7	6	5	4	3	2	1	0
EPROC			VERSION				

- **VERSION: Version of the Device**

- **EPROC: Embedded Processor**

EPROC			Processor
0	0	1	ARM946ES
0	1	0	ARM7TDMI
1	0	0	ARM920T
1	0	1	ARM926EJS
1	1	1	ARM926EJS + MAGIC DSP

- **NVPSIZ: Nonvolatile Program Memory Size**

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

- **NVPSIZ2 Second Nonvolatile Program Memory Size**

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

- **SRAMSIZ: Internal SRAM Size**

SRAMSIZ				Size
0	0	0	0	Reserved
0	0	0	1	1K bytes
0	0	1	0	2K bytes
0	0	1	1	48K bytes
0	1	0	0	112K bytes
0	1	0	1	4K bytes
0	1	1	0	80K bytes
0	1	1	1	160K bytes
1	0	0	0	8K bytes
1	0	0	1	16K bytes
1	0	1	0	32K bytes
1	0	1	1	64K bytes
1	1	0	0	128K bytes
1	1	0	1	256K bytes
1	1	1	0	96K bytes
1	1	1	1	512K bytes

- **ARCH: Architecture Identifier**

ARCH		
Hex	**Bin**	**Architecture**
0x19	0001 1001	AT91SAM9xx Series
0x29	0010 1001	AT91SAM9XExx Series
0x34	0011 0100	AT91x34 Series
0x37	0011 0111	CAP7 Series
0x39	0011 1001	CAP9 Series
0x3B	0011 1011	CAP11 Series
0x40	0100 0000	AT91x40 Series
0x42	0100 0010	AT91x42 Series
0x55	0101 0101	AT91x55 Series
0x60	0110 0000	AT91SAM7Axx Series
0x61	0110 0001	AT91SAM7AQxx Series
0x63	0110 0011	AT91x63 Series
0x70	0111 0000	AT91SAM7Sxx Series
0x71	0111 0001	AT91SAM7XCxx Series
0x72	0111 0010	AT91SAM7SExx Series
0x73	0111 0011	AT91SAM7Lxx Series
0x75	0111 0101	AT91SAM7Xxx Series
0x92	1001 0010	AT91x92 Series
0xE0	1001 0010	AT572 Series
0xF0	1111 0000	AT75Cxx Series

- **NVPTYP: Nonvolatile Program Memory Type**

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

- **EXT: Extension Flag**

0 = Chip ID has a single register definition without extension

1 = An extended Chip ID exists.

22.5.11 Debug Unit Chip ID Extension Register

Name: DBGU_EXID

Access Type: Read-only

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

- **EXID: Chip ID Extension**

Reads 0 if the bit EXT in DBGU_CIDR is 0.

22.5.12 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 power_on_reset signal.

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

23. Parallel Input/Output Controller (PIO)

23.1 Description

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

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

Each I/O line of the PIO Controller features:

- An input change interrupt enabling level change detection on any I/O line.
- A glitch filter providing rejection of pulses lower than one-half of clock cycle.
- Multi-drive capability similar to an open drain I/O line.
- Control of the pull-up of the I/O line.
- Input visibility and output control.

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

23.2 Block Diagram

Figure 23-1. Block Diagram

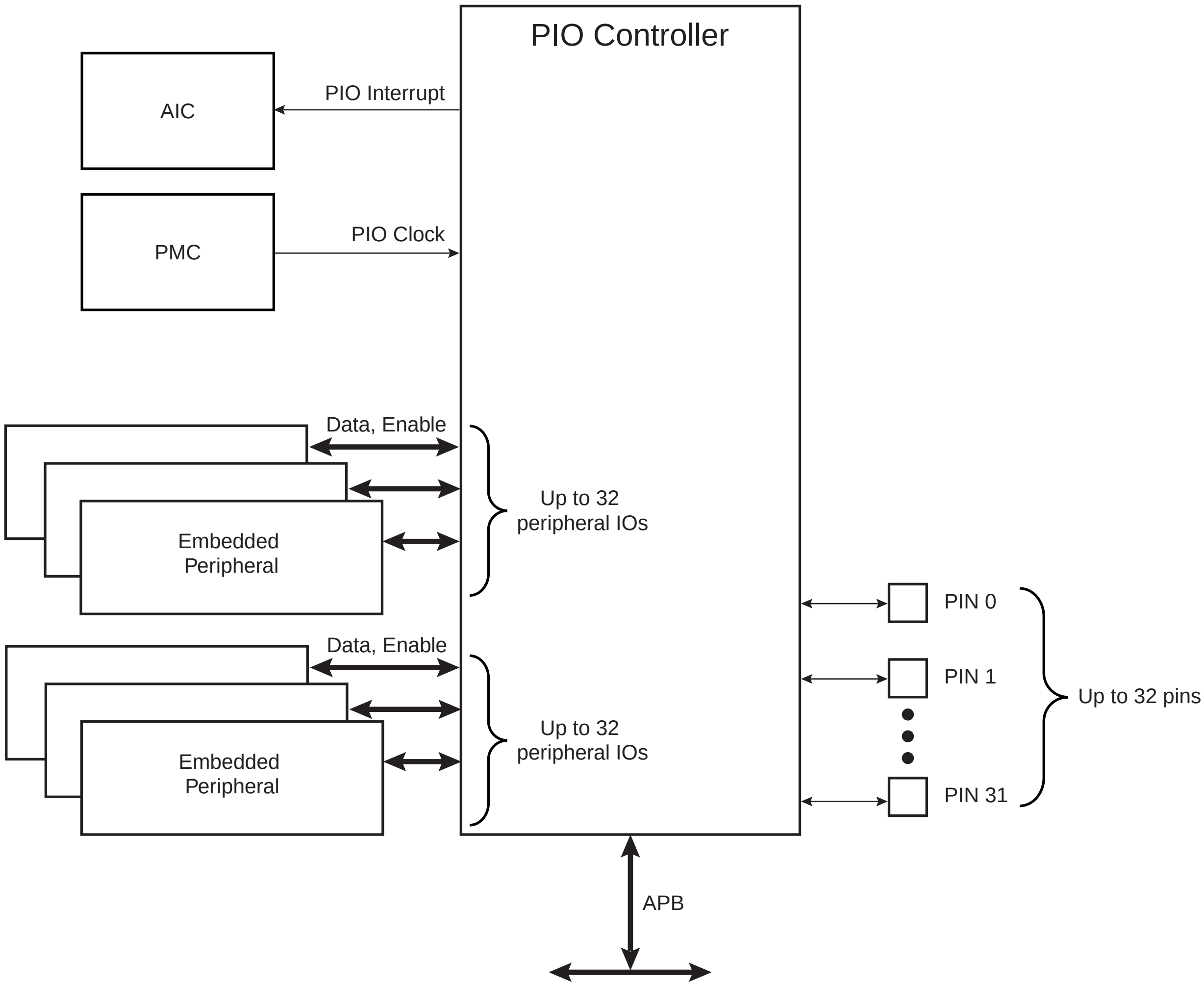
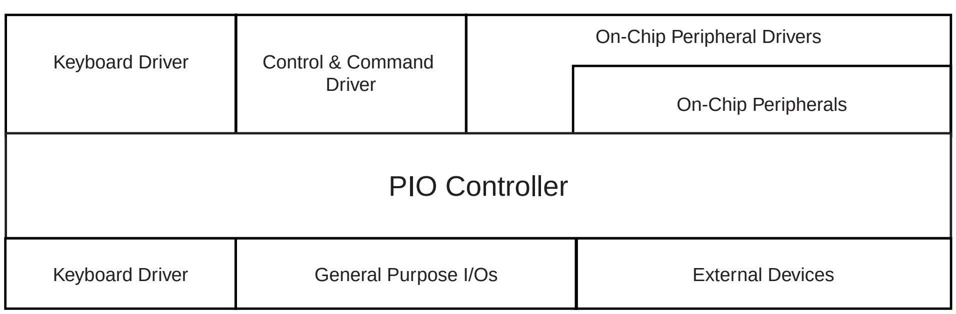


Figure 23-2. Application Block Diagram



23.3 Product Dependencies

23.3.1 Pin Multiplexing

Each pin is configurable, according to product definition as either a general-purpose I/O line only, or as an I/O line multiplexed with one or two peripheral I/Os. As the multiplexing is hardware-defined and thus product-dependent, the hardware designer and programmer must carefully determine the configuration of the PIO controllers required by their application. When an I/O line is general-purpose only, i.e. not multiplexed with any peripheral I/O, programming of the PIO Controller regarding the assignment to a peripheral has no effect and only the PIO Controller can control how the pin is driven by the product.

23.3.2 External Interrupt Lines

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

23.3.3 Power Management

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

However, when the clock is disabled, not all of the features of the PIO Controller are available. Note that the Input Change Interrupt and the read of the pin level require the clock to be validated.

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

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

23.3.4 Interrupt Generation

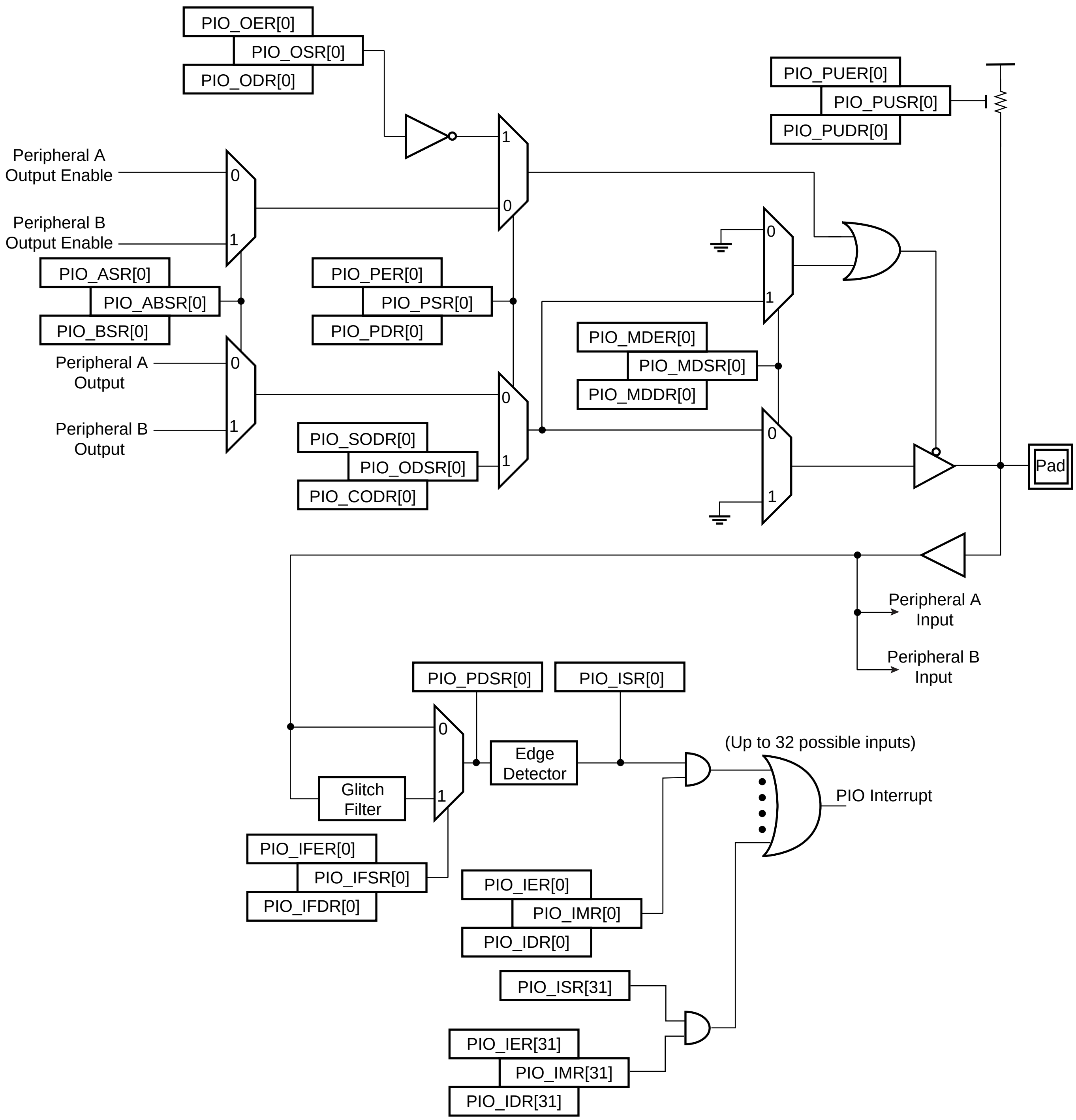
For interrupt handling, the PIO Controllers are considered as user peripherals. This means that the PIO Controller interrupt lines are connected among the interrupt sources 2 to 31. Refer to the PIO Controller peripheral identifier in the product description to identify the interrupt sources dedicated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

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

23.4 Functional Description

The PIO Controller features up to 32 fully-programmable I/O lines. Most of the control logic associated to each I/O is represented in Figure 23-3. In this description each signal shown represents but one of up to 32 possible indexes.

Figure 23-3. I/O Line Control Logic



23.4.1 Pull-up Resistor Control

Each I/O line is designed with an embedded pull-up resistor. The pull-up resistor can be enabled or disabled by writing respectively PIO_PUER (Pull-up Enable Register) and PIO_PUDR (Pull-up Disable 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.

23.4.2 I/O Line or Peripheral Function Selection

When a pin is multiplexed with one or two peripheral functions, the selection is controlled with the registers PIO_PER (PIO Enable Register) and PIO_PDR (PIO Disable Register). The register PIO_PSR (PIO Status Register) is the result of the set and clear registers and indicates whether the pin is controlled by the corresponding peripheral or by the PIO Controller. A value of 0 indicates that the pin is controlled by the corresponding on-chip peripheral selected in the PIO_ABSR (AB Select Status Register). A value of 1 indicates the pin is controlled by the PIO controller.

If a pin is used as a general purpose I/O line (not multiplexed with an on-chip peripheral), PIO_PER and PIO_PDR have no effect and PIO_PSR returns 1 for the corresponding bit.

After reset, most generally, the I/O lines are controlled by the PIO controller, i.e. PIO_PSR resets at 1. However, in some events, it is important that PIO lines are controlled by the peripheral (as in the case of memory chip select lines that must be driven inactive after reset or for address lines that must be driven low for booting out of an external memory). Thus, the reset value of PIO_PSR is defined at the product level, depending on the multiplexing of the device.

23.4.3 Peripheral A or B Selection

The PIO Controller provides multiplexing of up to two peripheral functions on a single pin. The selection is performed by writing PIO_ASR (A Select Register) and PIO_BSR (Select B Register). PIO_ABSR (AB Select Status Register) indicates which peripheral line is currently selected. For each pin, the corresponding bit at level 0 means peripheral A is selected whereas the corresponding bit at level 1 indicates that peripheral B is selected.

Note that multiplexing of peripheral lines A and B only affects the output line. The peripheral input lines are always connected to the pin input.

After reset, PIO_ABSR is 0, thus indicating that all the PIO lines are configured on peripheral A. However, peripheral A generally does not drive the pin as the PIO Controller resets in I/O line mode.

Writing in PIO_ASR and PIO_BSR manages PIO_ABSR regardless of the configuration of the pin. However, assignment of a pin to a peripheral function requires a write in the corresponding peripheral selection register (PIO_ASR or PIO_BSR) in addition to a write in PIO_PDR.

23.4.4 Output Control

When the I/0 line is assigned to a peripheral function, i.e. the corresponding bit in PIO_PSR is at 0, the drive of the I/O line is controlled by the peripheral. Peripheral A or B, depending on the value in PIO_ABSR, determines whether the pin is driven or not.

When the I/O line is controlled by the PIO controller, the pin can be configured to be driven. This is done by writing PIO_OER (Output Enable Register) and PIO_ODR (Output Disable Register).

The results of these write operations are detected in PIO_OSR (Output Status Register). When a bit in this register is at 0, the corresponding I/O line is used as an input only. When the bit is at 1, the corresponding I/O line is driven by the PIO controller.

The level driven on an I/O line can be determined by writing in PIO_SODR (Set Output Data Register) and PIO_CODR (Clear Output Data Register). These write operations respectively set and clear PIO_ODSR (Output Data Status Register), which represents the data driven on the I/O lines. Writing in PIO_OER and PIO_ODR manages PIO_OSR whether the pin is configured to be controlled by the PIO controller or assigned to a peripheral function. This enables configuration of the I/O line prior to setting it to be managed by the PIO Controller.

Similarly, writing in PIO_SODR and PIO_CODR effects PIO_ODSR. This is important as it defines the first level driven on the I/O line.

23.4.5 Synchronous Data Output

Controlling all parallel busses using several PIOs requires two successive write operations in the PIO_SODR and PIO_CODR registers. This may lead to unexpected transient values. The PIO controller offers a direct control of PIO outputs by single write access to PIO_ODSR (Output Data Status Register). Only bits unmasked by PIO_OWSR (Output Write Status Register) are written. The mask bits in the PIO_OWSR are set by writing to PIO_OWER (Output Write Enable Register) and cleared by writing to PIO_OWDR (Output Write Disable Register).

After reset, the synchronous data output is disabled on all the I/O lines as PIO_OWSR resets at 0x0.

23.4.6 Multi Drive Control (Open Drain)

Each I/O can be independently programmed in Open Drain by using the Multi Drive feature. This feature permits several drivers to be connected on the I/O line which is driven low only by each device. An external pull-up resistor (or enabling of the internal one) is generally required to guarantee a high level on the line.

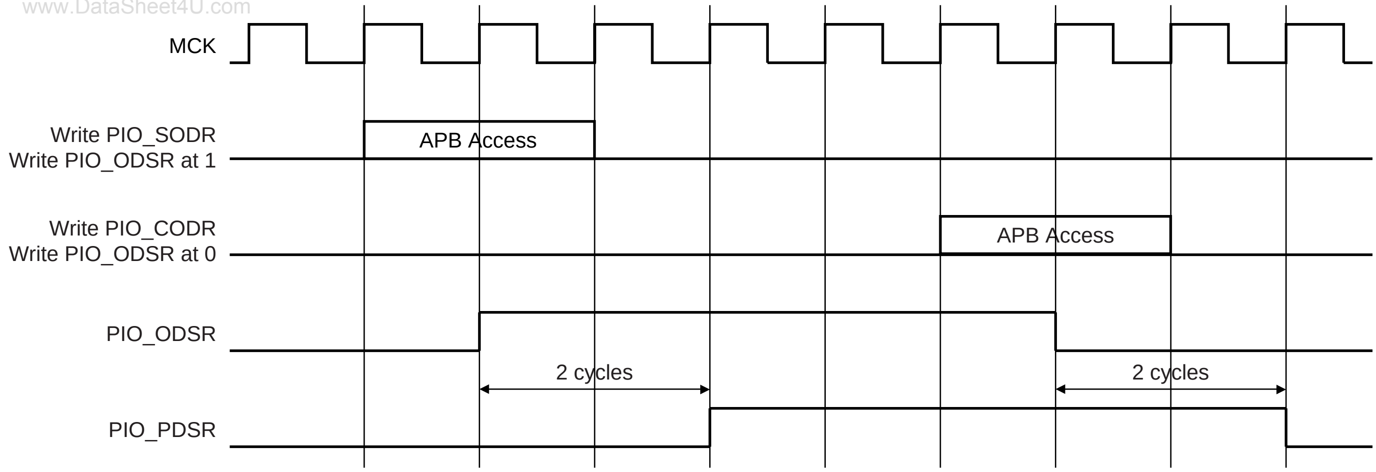
The Multi Drive feature is controlled by PIO_MDER (Multi-driver Enable Register) and PIO_MDDR (Multi-driver Disable Register). The Multi Drive can be selected whether the I/O line is controlled by the PIO controller or assigned to a peripheral function. PIO_MDSR (Multi-driver Status Register) indicates the pins that are configured to support external drivers.

After reset, the Multi Drive feature is disabled on all pins, i.e. PIO_MDSR resets at value 0x0.

23.4.7 Output Line Timings

Figure 23-4 shows how the outputs are driven either by writing PIO_SODR or PIO_CODR, or by directly writing PIO_ODSR. This last case is valid only if the corresponding bit in PIO_OWSR is set. Figure 23-4 also shows when the feedback in PIO_PDSR is available.

Figure 23-4. Output Line Timings



23.4.8 Inputs

The level on each I/O line can be read through PIO_PDSR (Pin Data Status Register). This register indicates the level of the I/O lines regardless of their configuration, whether uniquely as an input or driven by the PIO controller or driven by a peripheral.

Reading the I/O line levels requires the clock of the PIO controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.

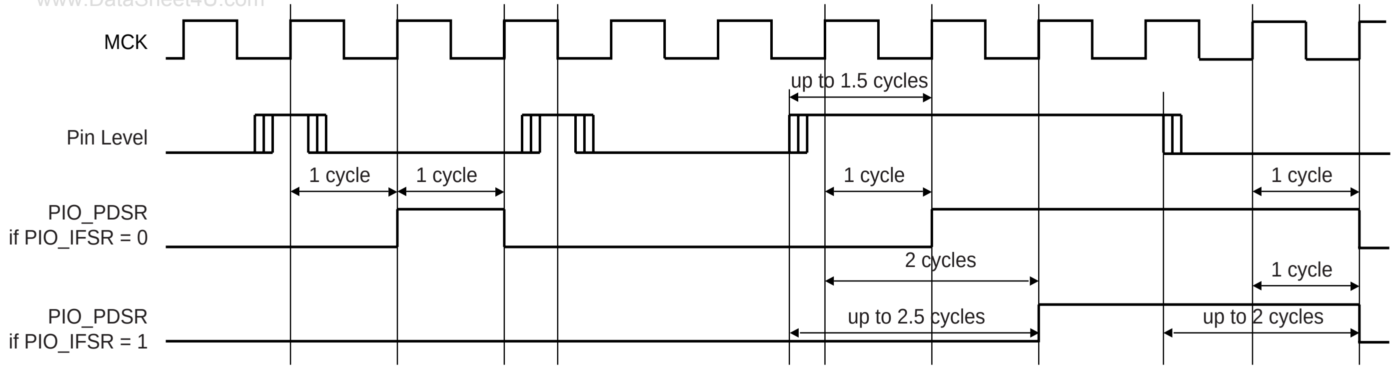
23.4.9 Input Glitch Filtering

Optional input glitch filters are independently programmable on each I/O line. When the glitch filter is enabled, a glitch with a duration of less than 1/2 Master Clock (MCK) cycle is automatically rejected, while a pulse with a duration of 1 Master Clock cycle or more is accepted. For pulse durations between 1/2 Master Clock cycle and 1 Master Clock cycle the pulse may or may not be taken into account, depending on the precise timing of its occurrence. Thus for a pulse to be visible it must exceed 1 Master Clock cycle, whereas for a glitch to be reliably filtered out, its duration must not exceed 1/2 Master Clock cycle. The filter introduces one Master Clock cycle latency if the pin level change occurs before a rising edge. However, this latency does not appear if the pin level change occurs before a falling edge. This is illustrated in Figure 23-5.

The glitch filters are controlled by the register set; PIO_IFER (Input Filter Enable Register), PIO_IFDR (Input Filter Disable Register) and PIO_IFSR (Input Filter Status Register). Writing PIO_IFER and PIO_IFDR respectively sets and clears bits in PIO_IFSR. This last register enables the glitch filter on the I/O lines.

When the glitch filter is enabled, it does not modify the behavior of the inputs on the peripherals. It acts only on the value read in PIO_PDSR and on the input change interrupt detection. The glitch filters require that the PIO Controller clock is enabled.

Figure 23-5. Input Glitch Filter Timing



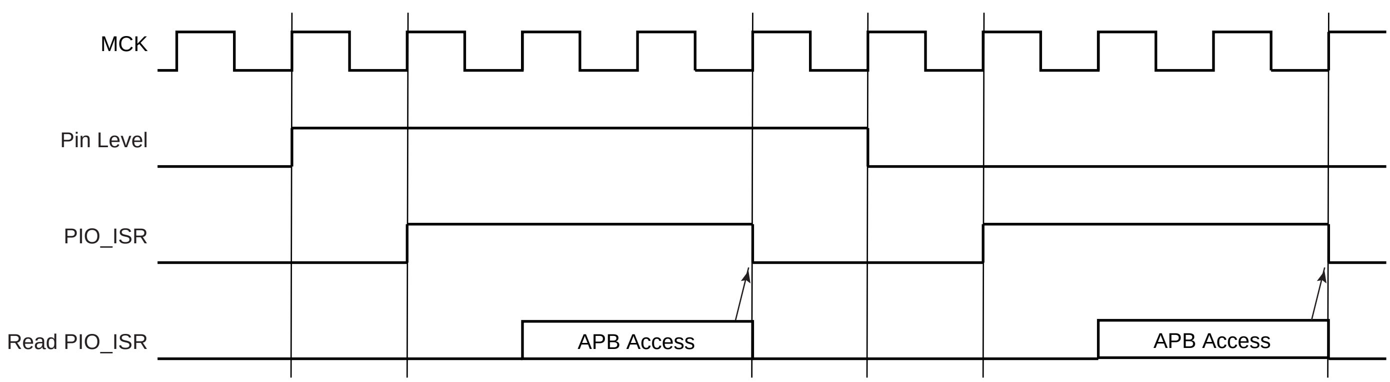
23.4.10 Input Change Interrupt

The PIO Controller can be programmed to generate an interrupt when it detects an input change on an I/O line. The Input Change Interrupt is controlled by writing PIO_IER (Interrupt Enable Register) and PIO_IDR (Interrupt Disable Register), which respectively enable and disable the input change interrupt by setting and clearing the corresponding bit in PIO_IMR (Interrupt Mask Register). As Input change detection is possible only by comparing two successive samplings of the input of the I/O line, the PIO Controller clock must be enabled. The Input Change Interrupt is available, regardless of the configuration of the I/O line, i.e. configured as an input only, controlled by the PIO Controller or assigned to a peripheral function.

When an input change is detected on an I/O line, the corresponding bit in PIO_ISR (Interrupt Status Register) is set. If the corresponding bit in PIO_IMR is set, the PIO Controller interrupt line is asserted. The interrupt signals of the thirty-two channels are ORed-wired together to generate a single interrupt signal to the Advanced Interrupt Controller.

When the software reads PIO_ISR, all the interrupts are automatically cleared. This signifies that all the interrupts that are pending when PIO_ISR is read must be handled.

Figure 23-6. Input Change Interrupt Timings



23.5 I/O Lines Programming Example

The programing example as shown in Table 23-1 below is used to define the following configuration.

- 4-bit output port on I/O lines 0 to 3, (should be written in a single write operation), open-drain, with pull-up resistor

- Four output signals on I/O lines 4 to 7 (to drive LEDs for example), driven high and low, no pull-up resistor
- Four input signals on I/O lines 8 to 11 (to read push-button states for example), with pull-up resistors, glitch filters and input change interrupts
- Four input signals on I/O line 12 to 15 to read an external device status (polled, thus no input change interrupt), no pull-up resistor, no glitch filter
- I/O lines 16 to 19 assigned to peripheral A functions with pull-up resistor
- I/O lines 20 to 23 assigned to peripheral B functions, no pull-up resistor
- I/O line 24 to 27 assigned to peripheral A with Input Change Interrupt and pull-up resistor

Table 23-1. Programming Example

Register	Value to be Written
PIO_PER	0x0000 FFFF
PIO_PDR	0x0FFF 0000
PIO_OER	0x0000 00FF
PIO_ODR	0x0FFF FF00
PIO_IFER	0x0000 0F00
PIO_IFDR	0x0FFF F0FF
PIO_SODR	0x0000 0000
PIO_CODR	0x0FFF FFFF
PIO_IER	0x0F00 0F00
PIO_IDR	0x00FF F0FF
PIO_MDER	0x0000 000F
PIO_MDDR	0x0FFF FFF0
PIO_PUDR	0x00F0 00F0
PIO_PUER	0x0F0F FF0F
PIO_ASR	0x0F0F 0000
PIO_BSR	0x00F0 0000
PIO_OWER	0x0000 000F
PIO_OWDR	0x0FFF FFF0

23.6 User Interface

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

Table 23-2. Register Mapping

Offset	Register	Name	Access	Reset Value
0x0000	PIO Enable Register	PIO_PER	Write-only	–
0x0004	PIO Disable Register	PIO_PDR	Write-only	–
0x0008	PIO Status Register	PIO_PSR	Read-only	(1)
0x000C	Reserved			
0x0010	Output Enable Register	PIO_OER	Write-only	–
0x0014	Output Disable Register	PIO_ODR	Write-only	–
0x0018	Output Status Register	PIO_OSR	Read-only	0x0000 0000
0x001C	Reserved			
0x0020	Glitch Input Filter Enable Register	PIO_IFER	Write-only	–
0x0024	Glitch Input Filter Disable Register	PIO_IFDR	Write-only	–
0x0028	Glitch Input Filter Status Register	PIO_IFSR	Read-only	0x0000 0000
0x002C	Reserved			
0x0030	Set Output Data Register	PIO_SODR	Write-only	–
0x0034	Clear Output Data Register	PIO_CODR	Write-only	
0x0038	Output Data Status Register	PIO_ODSR	Read-only or[2] Read/Write	–
0x003C	Pin Data Status Register	PIO_PDSR	Read-only	(3)
0x0040	Interrupt Enable Register	PIO_IER	Write-only	–
0x0044	Interrupt Disable Register	PIO_IDR	Write-only	–
0x0048	Interrupt Mask Register	PIO_IMR	Read-only	0x00000000
0x004C	Interrupt Status Register[4]	PIO_ISR	Read-only	0x00000000
0x0050	Multi-driver Enable Register	PIO_MDER	Write-only	–
0x0054	Multi-driver Disable Register	PIO_MDDR	Write-only	–
0x0058	Multi-driver Status Register	PIO_MDSR	Read-only	0x00000000
0x005C	Reserved			
0x0060	Pull-up Disable Register	PIO_PUDR	Write-only	–
0x0064	Pull-up Enable Register	PIO_PUER	Write-only	–
0x0068	Pad Pull-up Status Register	PIO_PUSR	Read-only	0x00000000
0x006C	Reserved			

Table 23-2. Register Mapping (Continued)

Offset	Register	Name	Access	Reset Value
0x0070	Peripheral A Select Register[(5)]	PIO_ASR	Write-only	–
0x0074	Peripheral B Select Register[(5)]	PIO_BSR	Write-only	–
0x0078	AB Status Register[(5)]	PIO_ABSR	Read-only	0x00000000
0x007C to 0x009C	Reserved			
0x00A0	Output Write Enable	PIO_OWER	Write-only	–
0x00A4	Output Write Disable	PIO_OWDR	Write-only	–
0x00A8	Output Write Status Register	PIO_OWSR	Read-only	0x00000000
0x00AC	Reserved			

Notes:
1. Reset value of PIO_PSR depends on the product implementation.
2. PIO_ODSR is Read-only or Read/Write depending on PIO_OWSR I/O lines.
3. Reset value of PIO_PDSR depends on the level of the I/O lines. Reading the I/O line levels requires the clock of the PIO Controller to be enabled, otherwise PIO_PDSR reads the levels present on the I/O line at the time the clock was disabled.
4. PIO_ISR is reset at 0x0. However, the first read of the register may read a different value as input changes may have occurred.
5. 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.

23.6.1 PIO Controller PIO Enable Register

Name: PIO_PER

Access Type: Write-only

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

- **P0-P31: PIO Enable**

0 = No effect.

1 = Enables the PIO to control the corresponding pin (disables peripheral control of the pin).

23.6.2 PIO Controller PIO Disable Register

Name: PIO_PDR

Access Type: Write-only

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

- **P0-P31: PIO Disable**

0 = No effect.

1 = Disables the PIO from controlling the corresponding pin (enables peripheral control of the pin).

23.6.3 PIO Controller PIO Status Register

Name: PIO_PSR

Access Type: Read-only

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

- **P0-P31: PIO Status**

0 = PIO is inactive on the corresponding I/O line (peripheral is active).

1 = PIO is active on the corresponding I/O line (peripheral is inactive).

23.6.4 PIO Controller Output Enable Register

Name: PIO_OER

Access Type: Write-only

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

- **P0-P31: Output Enable**

0 = No effect.

1 = Enables the output on the I/O line.

23.6.5 PIO Controller Output Disable Register

Name: PIO_ODR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Disable**

0 = No effect.

1 = Disables the output on the I/O line.

23.6.6 PIO Controller Output Status Register

Name: PIO_OSR

Access Type: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Status**

0 = The I/O line is a pure input.

1 = The I/O line is enabled in output.

23.6.7 PIO Controller Input Filter Enable Register

Name: PIO_IFER

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Filter Enable**

0 = No effect.

1 = Enables the input glitch filter on the I/O line.

23.6.8 PIO Controller Input Filter Disable Register

Name: PIO_IFDR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input Filter Disable**

0 = No effect.

1 = Disables the input glitch filter on the I/O line.

23.6.9 PIO Controller Input Filter Status Register

Name: PIO_IFSR

Access Type: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Input 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.

23.6.10 PIO Controller Set Output Data Register

Name: PIO_SODR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Set Output Data**

0 = No effect.

1 = Sets the data to be driven on the I/O line.

23.6.11 PIO Controller Clear Output Data Register

Name: PIO_CODR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Set Output Data**

0 = No effect.

1 = Clears the data to be driven on the I/O line.

23.6.12 PIO Controller Output Data Status Register

Name: PIO_ODSR

Access Type: Read-only or Read/Write

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Data Status**

0 = The data to be driven on the I/O line is 0.

1 = The data to be driven on the I/O line is 1.

23.6.13 PIO Controller Pin Data Status Register

Name: PIO_PDSR

Access Type: Read-only

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

- **P0-P31: Output Data Status**

0 = The I/O line is at level 0.

1 = The I/O line is at level 1.

23.6.14 PIO Controller Interrupt Enable Register

Name: PIO_IER

Access Type: Write-only

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

- **P0-P31: Input Change Interrupt Enable**

0 = No effect.

1 = Enables the Input Change Interrupt on the I/O line.

23.6.15 PIO Controller Interrupt Disable Register

Name: PIO_IDR

Access Type: Write-only

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

- **P0-P31: Input Change Interrupt Disable**

0 = No effect.

1 = Disables the Input Change Interrupt on the I/O line.

23.6.16 PIO Controller Interrupt Mask Register

Name: PIO_IMR

Access Type: Read-only

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

- **P0-P31: Input Change Interrupt Mask**

0 = Input Change Interrupt is disabled on the I/O line.

1 = Input Change Interrupt is enabled on the I/O line.

23.6.17 PIO Controller Interrupt Status Register

Name: PIO_ISR

Access Type: Read-only

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

- **P0-P31: Input Change Interrupt Status**

0 = No Input Change has been detected on the I/O line since PIO_ISR was last read or since reset.

1 = At least one Input Change has been detected on the I/O line since PIO_ISR was last read or since reset.

23.6.18 PIO Multi-driver Enable Register

Name: PIO_MDER

Access Type: Write-only

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

- **P0-P31: Multi Drive Enable.**

0 = No effect.

1 = Enables Multi Drive on the I/O line.

23.6.19 PIO Multi-driver Disable Register

Name: PIO_MDDR

Access Type: Write-only

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

- **P0-P31: Multi Drive Disable.**

0 = No effect.

1 = Disables Multi Drive on the I/O line.

23.6.20 PIO Multi-driver Status Register

Name: PIO_MDSR

Access Type: Read-only

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

- **P0-P31: Multi Drive Status.**

0 = The Multi Drive is disabled on the I/O line. The pin is driven at high and low level.

1 = The Multi Drive is enabled on the I/O line. The pin is driven at low level only.

23.6.21 PIO Pull Up Disable Register

Name: PIO_PUDR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Pull Up Disable.**

0 = No effect.

1 = Disables the pull up resistor on the I/O line.

23.6.22 PIO Pull Up Enable Register

Name: PIO_PUER

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Pull Up Enable.**

0 = No effect.

1 = Enables the pull up resistor on the I/O line.

23.6.23 PIO Pull Up Status Register

Name: PIO_PUSR

Access Type: Read-only

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

- **P0-P31: Pull Up Status.**

0 = Pull Up resistor is enabled on the I/O line.

1 = Pull Up resistor is disabled on the I/O line.

23.6.24 PIO Peripheral A Select Register

Name: PIO_ASR

Access Type: Write-only

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

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

0 = No effect.

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

23.6.25 PIO Peripheral B Select Register

Name: PIO_BSR

Access Type: Write-only

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

- **P0-P31: Peripheral B Select.**

0 = No effect.

1 = Assigns the I/O line to the peripheral B function.

23.6.26 PIO Peripheral A B Status Register

Name: PIO_ABSR

Access Type: Read-only

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

- **P0-P31: Peripheral A B Status.**

0 = The I/O line is assigned to the Peripheral A.

1 = The I/O line is assigned to the Peripheral B.

23.6.27 PIO Output Write Enable Register

Name: PIO_OWER

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Write Enable.**

0 = No effect.

1 = Enables writing PIO_ODSR for the I/O line.

23.6.28 PIO Output Write Disable Register

Name: PIO_OWDR

Access Type: Write-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Write Disable.**

0 = No effect.

1 = Disables writing PIO_ODSR for the I/O line.

23.6.29 PIO Output Write Status Register

Name: PIO_OWSR

Access Type: Read-only

31	30	29	28	27	26	25	24
P31	P30	P29	P28	P27	P26	P25	P24

23	22	21	20	19	18	17	16
P23	P22	P21	P20	P19	P18	P17	P16

15	14	13	12	11	10	9	8
P15	P14	P13	P12	P11	P10	P9	P8

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

- **P0-P31: Output Write Status.**

0 = Writing PIO_ODSR does not affect the I/O line.

1 = Writing PIO_ODSR affects the I/O line.

24. Serial Peripheral Interface (SPI)

24.1 Description

The Serial Peripheral Interface (SPI) circuit is a synchronous serial data link that provides communication with external devices in Master or Slave Mode. It also enables communication between processors if an external processor is connected to the system.

The Serial Peripheral Interface is essentially a shift register that serially transmits data bits to other SPIs. During a data transfer, one SPI system acts as the "master"' which controls the data flow, while the other devices act as "slaves'' which have data shifted into and out by the master. Different CPUs can take turn being masters (Multiple Master Protocol opposite to Single Master Protocol where one CPU is always the master while all of the others are always slaves) and one master may simultaneously shift data into multiple slaves. However, only one slave may drive its output to write data back to the master at any given time.

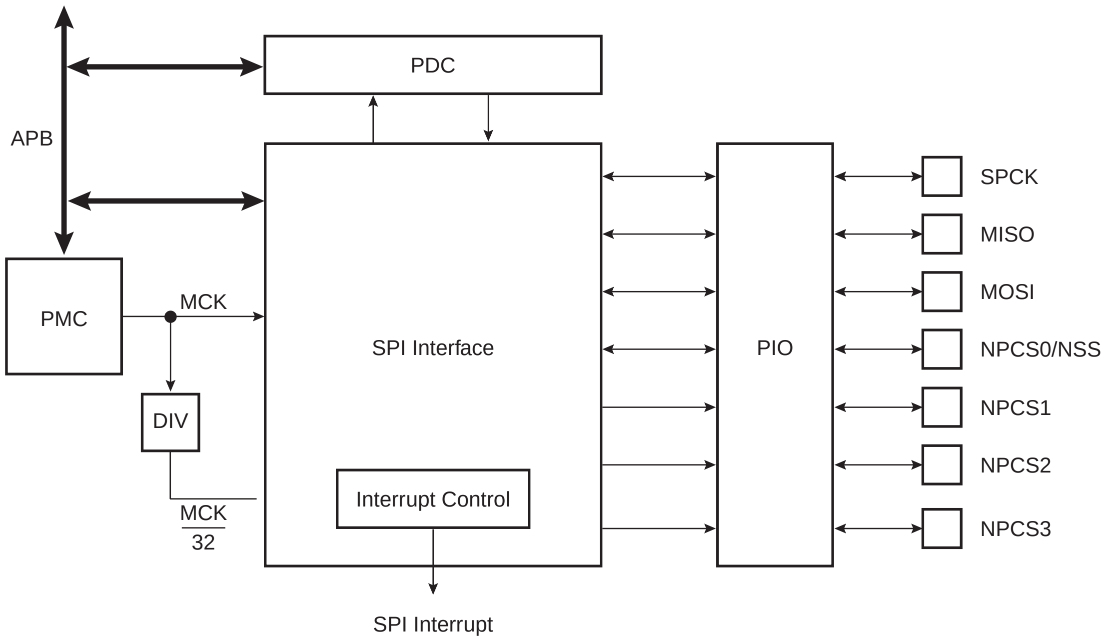
A slave device is selected when the master asserts its NSS signal. If multiple slave devices exist, the master generates a separate slave select signal for each slave (NPCS).

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

- Master Out Slave In (MOSI): This data line supplies the output data from the master shifted into the input(s) of the slave(s).
- Master In Slave Out (MISO): This data line supplies the output data from a slave to the input of the master. There may be no more than one slave transmitting data during any particular transfer.
- Serial Clock (SPCK): This control line is driven by the master and regulates the flow of the data bits. The master may transmit data at a variety of baud rates; the SPCK line cycles once for each bit that is transmitted.
- Slave Select (NSS): This control line allows slaves to be turned on and off by hardware.

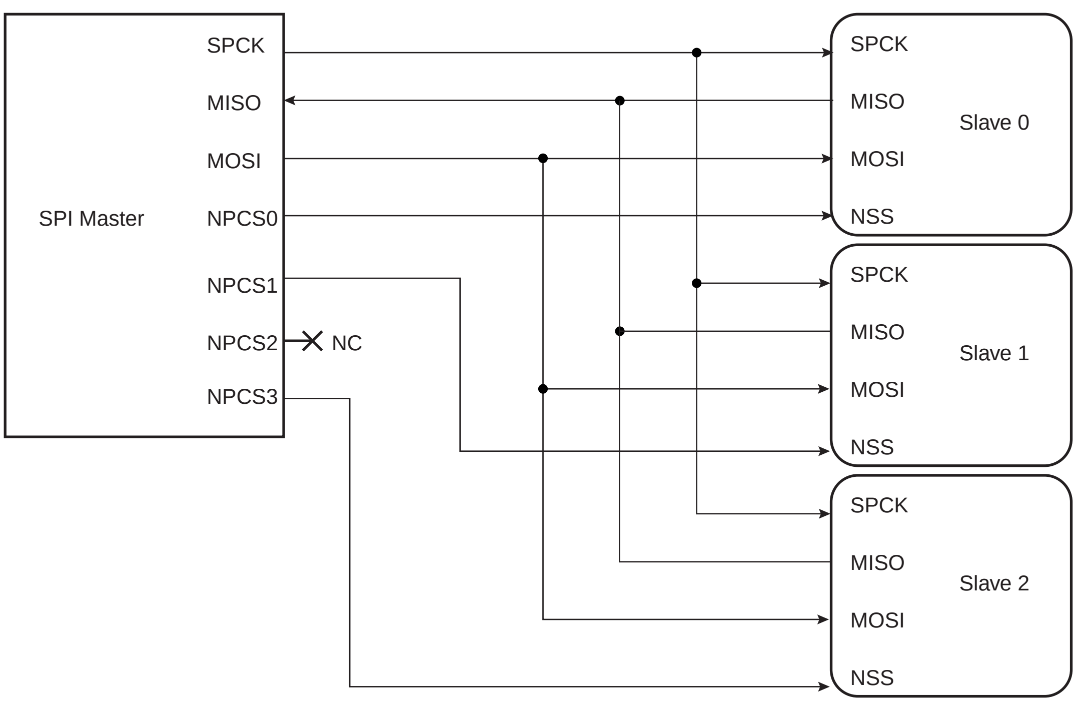
24.2 Block Diagram

Figure 24-1. Block Diagram



24.3 Application Block Diagram

Figure 24-2. Application Block Diagram: Single Master/Multiple Slave Implementation



24.4 Signal Description

Table 24-1. Signal Description

Pin Name	Pin Description	Type	
		Master	Slave
MISO	Master In Slave Out	Input	Output
MOSI	Master Out Slave In	Output	Input
SPCK	Serial Clock	Output	Input
NPCS1-NPCS3	Peripheral Chip Selects	Output	Unused
NPCS0/NSS	Peripheral Chip Select/Slave Select	Output	Input

24.5 Product Dependencies

24.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the SPI pins to their peripheral functions.

24.5.2 Power Management

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

24.5.3 Interrupt

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

24.6 Functional Description

24.6.1 Modes of Operation

The SPI operates in Master Mode or in Slave Mode.

Operation in Master Mode is programmed by writing at 1 the MSTR bit in the Mode Register. The pins NPCS0 to NPCS3 are all configured as outputs, the SPCK pin is driven, the MISO line is wired on the receiver input and the MOSI line driven as an output by the transmitter.

If the MSTR bit is written at 0, the SPI operates in Slave Mode. The MISO line is driven by the transmitter output, the MOSI line is wired on the receiver input, the SPCK pin is driven by the transmitter to synchronize the receiver. The NPCS0 pin becomes an input, and is used as a Slave Select signal (NSS). The pins NPCS1 to NPCS3 are not driven and can be used for other purposes.

The data transfers are identically programmable for both modes of operations. The baud rate generator is activated only in Master Mode.

24.6.2 Data Transfer

Four combinations of polarity and phase are available for data transfers. The clock polarity is programmed with the CPOL bit in the Chip Select Register. The clock phase is programmed with the NCPHA bit. These two parameters determine the edges of the clock signal on which data is driven and sampled. Each of the two parameters has two possible states, resulting in four possible combinations that are incompatible with one another. Thus, a master/slave pair must use the same parameter pair values to communicate. If multiple slaves are used and fixed in different configurations, the master must reconfigure itself each time it needs to communicate with a different slave.

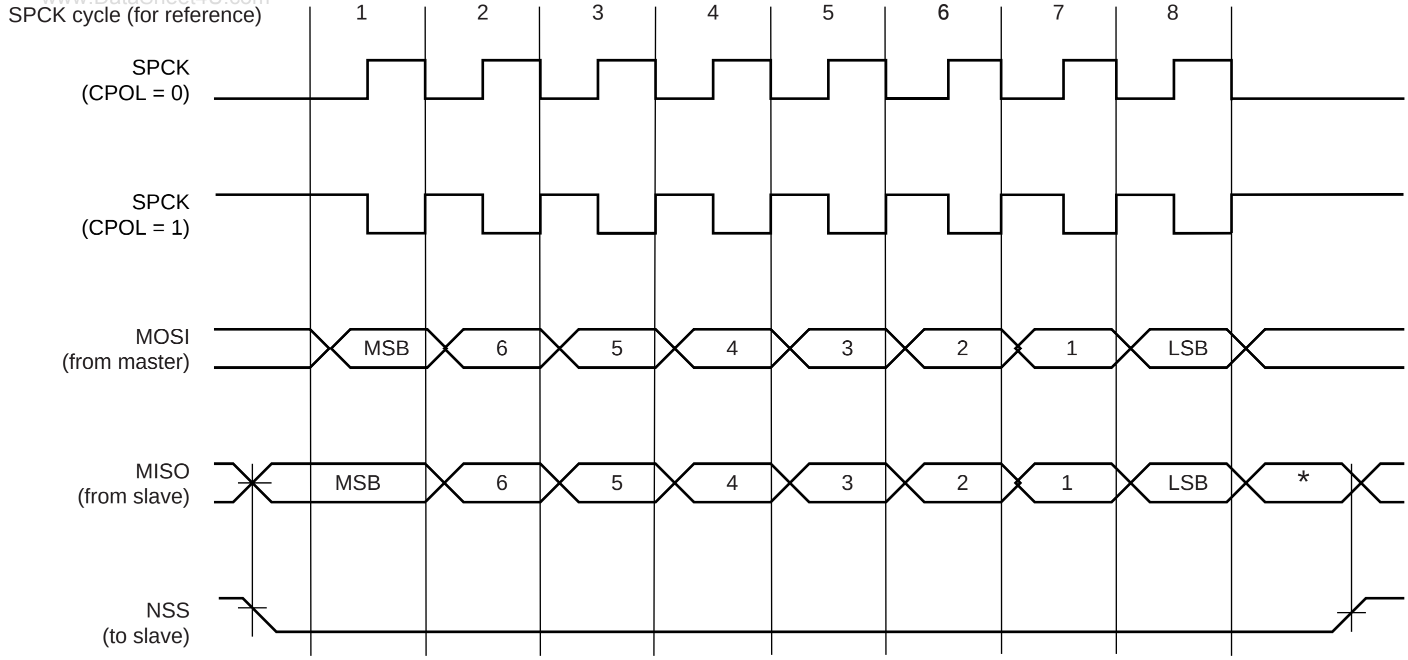
Table 24-2 shows the four modes and corresponding parameter settings.

Table 24-2. SPI Bus Protocol Mode

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

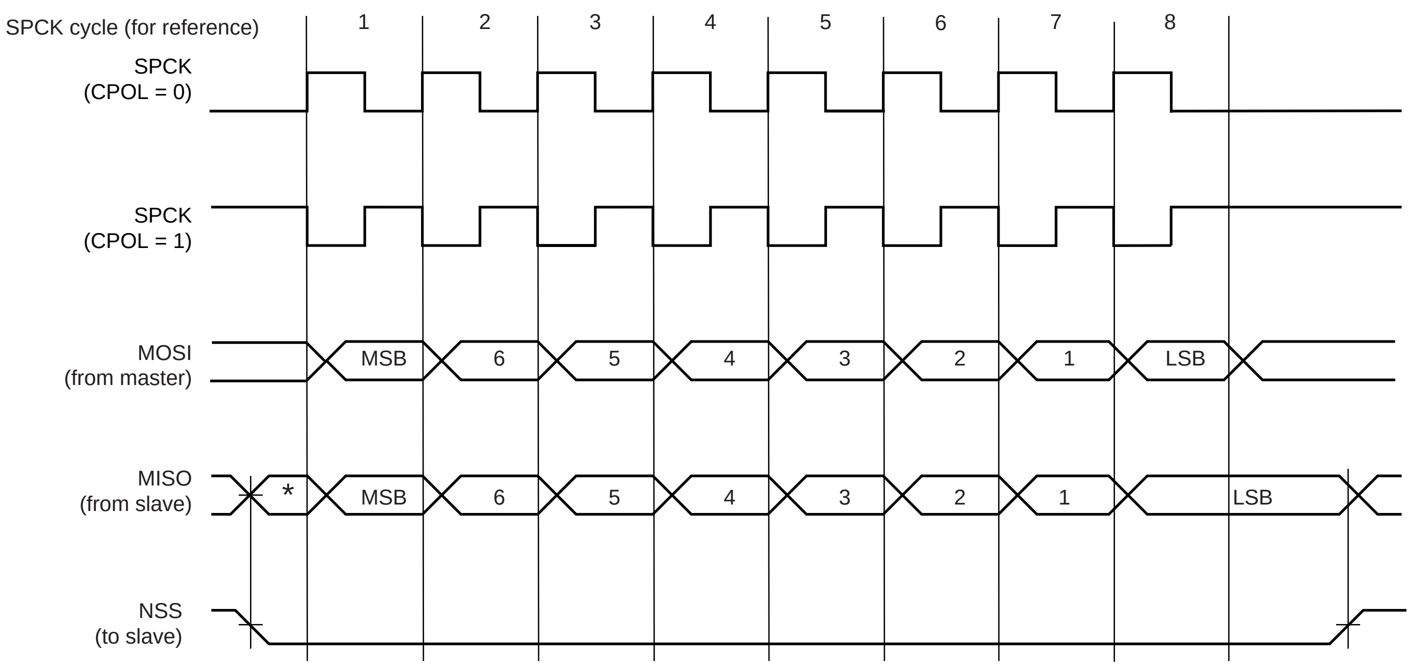
Figure 24-3 and Figure 24-4 show examples of data transfers.

Figure 24-3. SPI Transfer Format (NCPHA = 1, 8 bits per transfer)



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

Figure 24-4. SPI Transfer Format (NCPHA = 0, 8 bits per transfer)



* Not defined but normally LSB of previous character transmitted.

24.6.3 Master Mode Operations

When configured in Master Mode, the SPI operates on the clock generated by the internal programmable baud rate generator. It fully controls the data transfers to and from the slave(s) connected to the SPI bus. The SPI drives the chip select line to the slave and the serial clock signal (SPCK).

The SPI features two holding registers, the Transmit Data Register and the Receive Data Register, and a single Shift Register. The holding registers maintain the data flow at a constant rate.

After enabling the SPI, a data transfer begins when the processor writes to the SPI_TDR (Transmit Data Register). The written data is immediately transferred in the Shift Register and transfer on the SPI bus starts. While the data in the Shift Register is shifted on the MOSI line, the MISO line is sampled and shifted in the Shift Register. Transmission cannot occur without reception.

Before writing the TDR, the PCS field must be set in order to select a slave.

If new data is written in SPI_TDR during the transfer, it stays in it until the current transfer is completed. Then, the received data is transferred from the Shift Register to SPI_RDR, the data in SPI_TDR is loaded in the Shift Register and a new transfer starts.

The transfer of a data written in SPI_TDR in the Shift Register is indicated by the TDRE bit (Transmit Data Register Empty) in the Status Register (SPI_SR). When new data is written in SPI_TDR, this bit is cleared. The TDRE bit is used to trigger the Transmit PDC channel.

The end of transfer is indicated by the TXEMPTY flag in the SPI_SR register. If a transfer delay (DLYBCT) is greater than 0 for the last transfer, TXEMPTY is set after the completion of said delay. The master clock (MCK) can be switched off at this time.

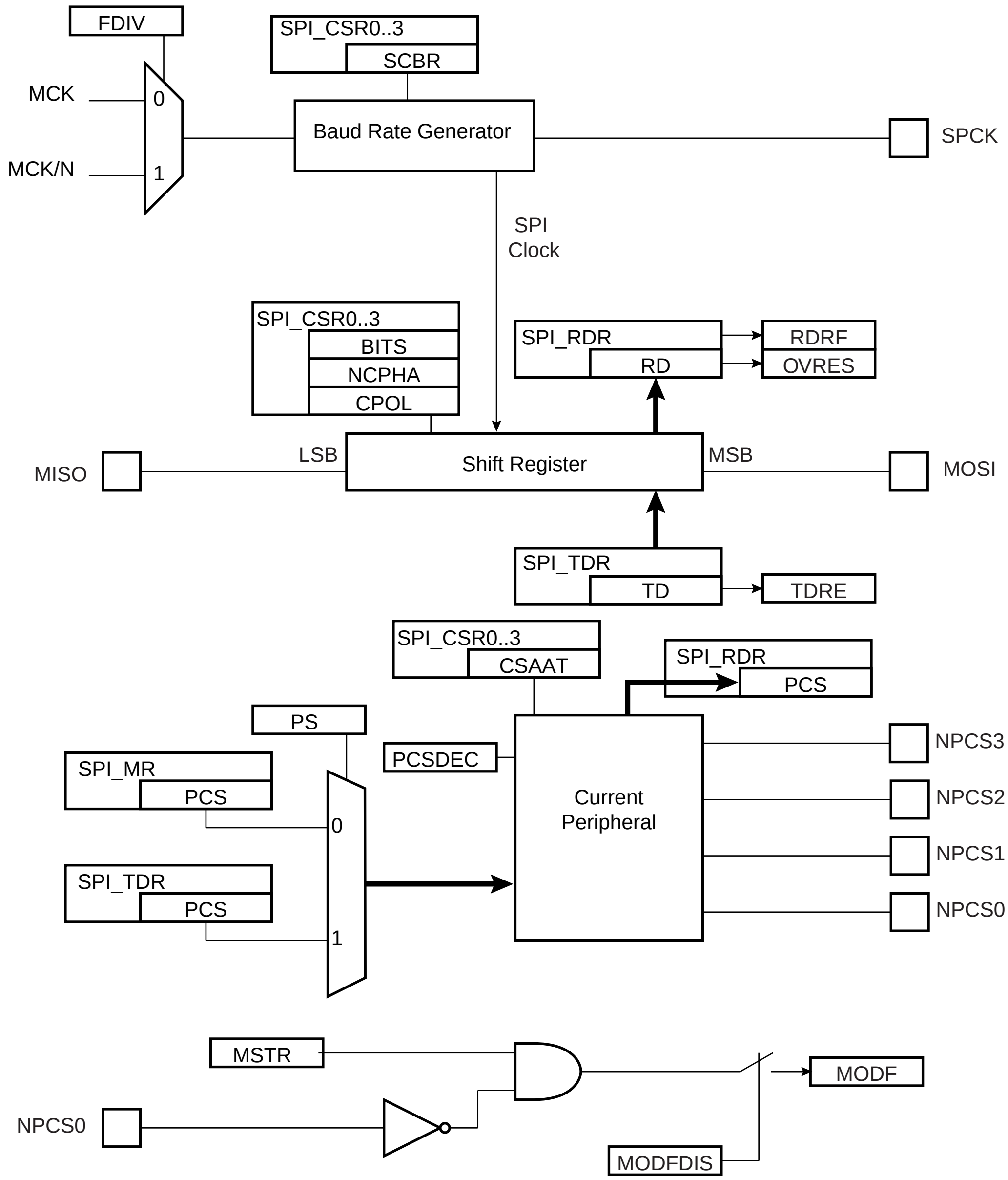
The transfer of received data from the Shift Register in SPI_RDR is indicated by the RDRF bit (Receive Data Register Full) in the Status Register (SPI_SR). When the received data is read, the RDRF bit is cleared.

If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI_SR is set. As long as this flag is set, data is loaded in SPI_RDR. The user has to read the status register to clear the OVRES bit.

Figure 24-6 on page 346 shows a flow chart describing how transfers are handled.

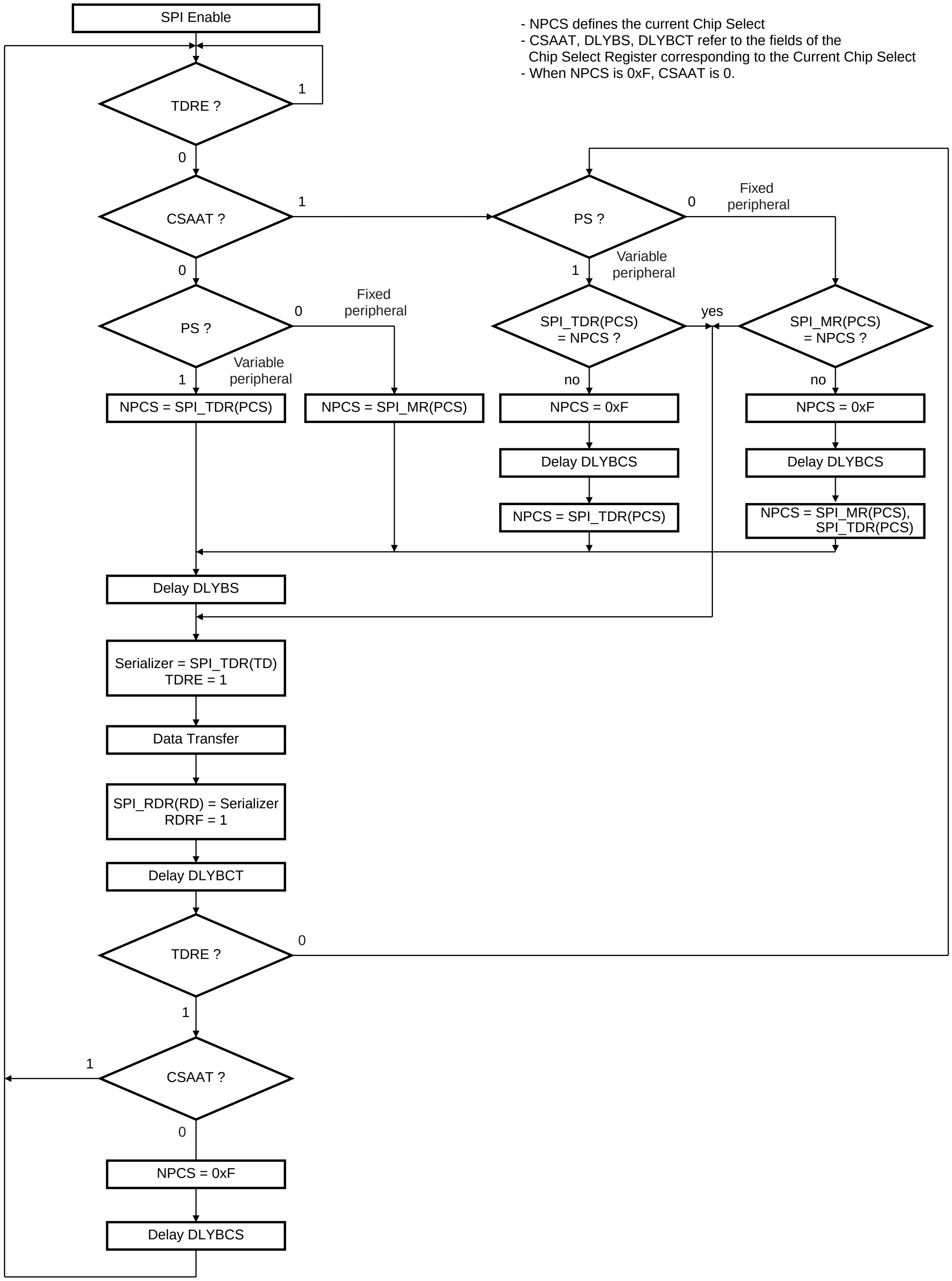
24.6.3.1 Master Mode Block Diagram

Figure 24-5. Master Mode Block Diagram



24.6.3.2 Master Mode Flow Diagram

Figure 24-6. Master Mode Flow Diagram S



24.6.3.3 Clock Generation

The SPI Baud rate clock is generated by dividing the Master Clock (MCK) or the Master Clock divided by 32, by a value between 1 and 255. The selection between Master Clock or Master Clock divided by 32 is done by the FDIV value set in the Mode Register

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

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

The divisor can be defined independently for each chip select, as it has to be programmed in the SCBR field of the Chip Select Registers. This allows the SPI to automatically adapt the baud rate for each interfaced peripheral without reprogramming.

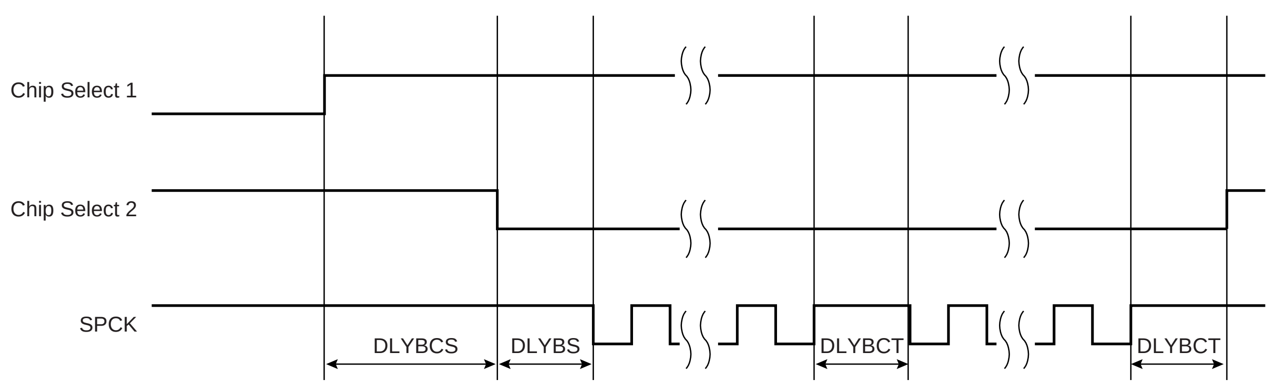
24.6.3.4 Transfer Delays

Figure 24-7 shows a chip select transfer change and consecutive transfers on the same chip select. Three delays can be programmed to modify the transfer waveforms:

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

These delays allow the SPI to be adapted to the interfaced peripherals and their speed and bus release time.

Figure 24-7. Programmable Delays



24.6.3.5 Peripheral Selection

The serial peripherals are selected through the assertion of the NPCS0 to NPCS3 signals. By default, all the NPCS signals are high before and after each transfer.

The peripheral selection can be performed in two different ways:

- Fixed Peripheral Select: SPI exchanges data with only one peripheral
- Variable Peripheral Select: Data can be exchanged with more than one peripheral

Fixed Peripheral Select is activated by writing the PS bit to zero in SPI_MR (Mode Register). In this case, the current peripheral is defined by the PCS field in SPI_MR and the PCS field in the SPI_TDR has no effect.

Variable Peripheral Select is activated by setting PS bit to one. The PCS field in SPI_TDR is used to select the current peripheral. This means that the peripheral selection can be defined for each new data.

The Fixed Peripheral Selection allows buffer transfers with a single peripheral. Using the PDC is an optimal means, as the size of the data transfer between the memory and the SPI is either 8 bits or 16 bits. However, changing the peripheral selection requires the Mode Register to be reprogrammed.

The Variable Peripheral Selection allows buffer transfers with multiple peripherals without reprogramming the Mode Register. Data written in SPI_TDR is 32 bits wide and defines the real data to be transmitted and the peripheral it is destined to. Using the PDC in this mode requires 32-bit wide buffers, with the data in the LSBs and the PCS and LASTXFER fields in the MSBs, however the SPI still controls the number of bits (8 to16) to be transferred through MISO and MOSI lines with the chip select configuration registers. This is not the optimal means in term of memory size for the buffers, but it provides a very effective means to exchange data with several peripherals without any intervention of the processor.

24.6.3.6 Peripheral Chip Select Decoding

The user can program the SPI to operate with up to 15 peripherals by decoding the four Chip Select lines, NPCS0 to NPCS3 with an external logic. This can be enabled by writing the PCSDEC bit at 1 in the Mode Register (SPI_MR).

When operating without decoding, the SPI makes sure that in any case only one chip select line is activated, i.e. driven low at a time. If two bits are defined low in a PCS field, only the lowest numbered chip select is driven low.

When operating with decoding, the SPI directly outputs the value defined by the PCS field of either the Mode Register or the Transmit Data Register (depending on PS).

As the SPI sets a default value of 0xF on the chip select lines (i.e. all chip select lines at 1) when not processing any transfer, only 15 peripherals can be decoded.

The SPI has only four Chip Select Registers, not 15. As a result, when decoding is activated, each chip select defines the characteristics of up to four peripherals. As an example, SPI_CRS0 defines the characteristics of the externally decoded peripherals 0 to 3, corresponding to the PCS values 0x0 to 0x3. Thus, the user has to make sure to connect compatible peripherals on the decoded chip select lines 0 to 3, 4 to 7, 8 to 11 and 12 to 14.

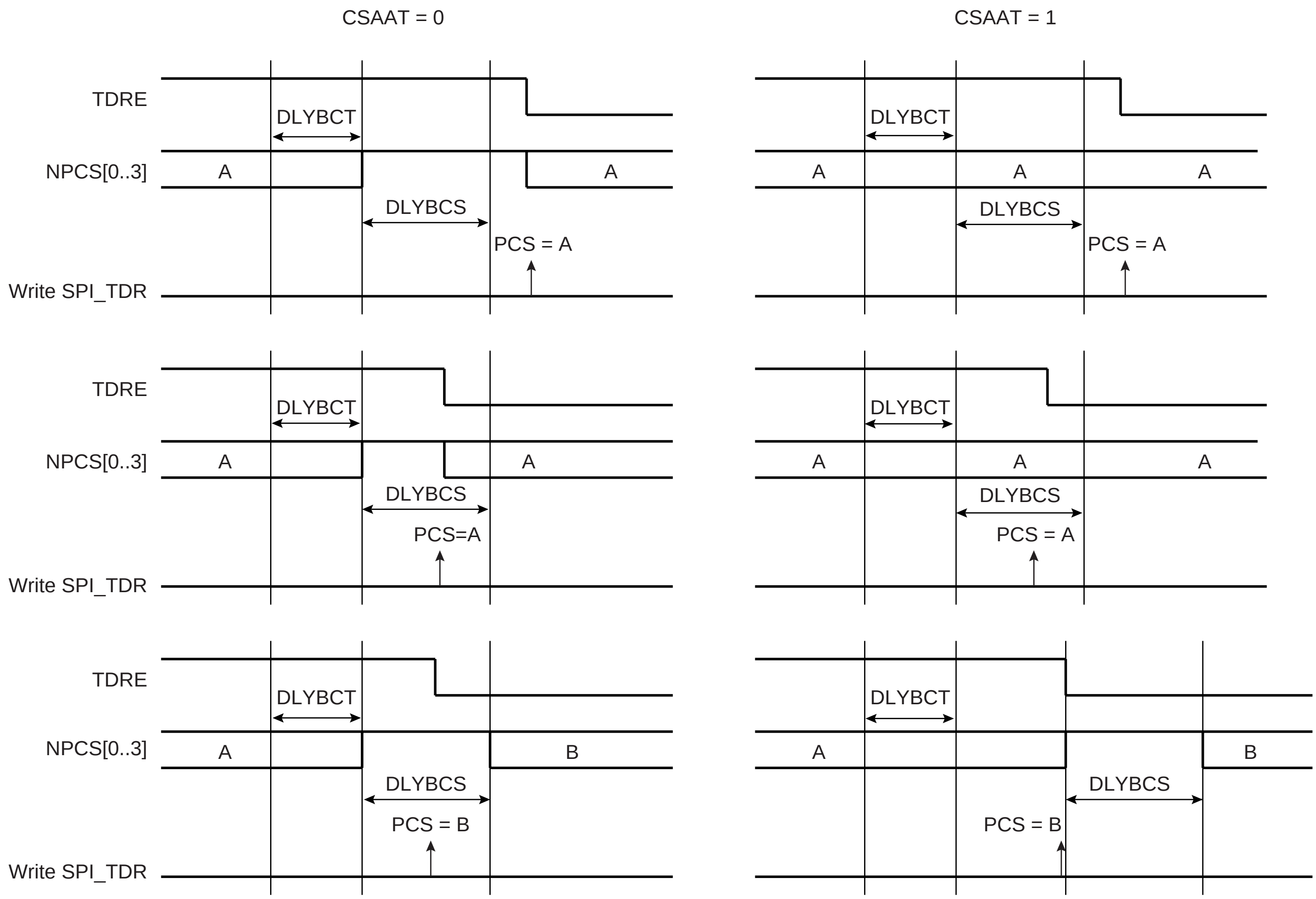
24.6.3.7 Peripheral Deselection

When operating normally, as soon as the transfer of the last data written in SPI_TDR is completed, the NPCS lines all rise. This might lead to runtime error if the processor is too long in responding to an interrupt, and thus might lead to difficulties for interfacing with some serial peripherals requiring the chip select line to remain active during a full set of transfers.

To facilitate interfacing with such devices, the Chip Select Register can be programmed with the CSAAT bit (Chip Select Active After Transfer) at 1. This allows the chip select lines to remain in their current state (low = active) until transfer to another peripheral is required.

Figure 24-8 shows different peripheral deselection cases and the effect of the CSAAT bit.

Figure 24-8. Peripheral Deselection



24.6.3.8 *Mode Fault Detection*

A mode fault is detected when the SPI is programmed in Master Mode and a low level is driven by an external master on the NPCS0/NSS signal. NPCS0, MOSI, MISO and SPCK must be configured in open drain through the PIO controller, so that external pull up resistors are needed to guarantee high level.

When a mode fault is detected, the MODF bit in the SPI_SR is set until the SPI_SR is read and the SPI is automatically disabled until re-enabled by writing the SPIEN bit in the SPI_CR (Control Register) at 1.

By default, the Mode Fault detection circuitry is enabled. The user can disable Mode Fault detection by setting the MODFDIS bit in the SPI Mode Register (SPI_MR).

24.6.4 SPI Slave Mode

When operating in Slave Mode, the SPI processes data bits on the clock provided on the SPI clock pin (SPCK).

The SPI waits for NSS to go active before receiving the serial clock from an external master. When NSS falls, the clock is validated on the serializer, which processes the number of bits defined by the BITS field of the Chip Select Register 0 (SPI_CSR0). These bits are processed following a phase and a polarity defined respectively by the NCPHA and CPOL bits of the SPI_CSR0. Note that BITS, CPOL and NCPHA of the other Chip Select Registers have no effect when the SPI is programmed in Slave Mode.

The bits are shifted out on the MISO line and sampled on the MOSI line.

When all the bits are processed, the received data is transferred in the Receive Data Register and the RDRF bit rises. If the SPI_RDR (Receive Data Register) has not been read before new data is received, the Overrun Error bit (OVRES) in SPI_SR is set. As long as this flag is set, data is loaded in SPI_RDR. The user has to read the status register to clear the OVRES bit.

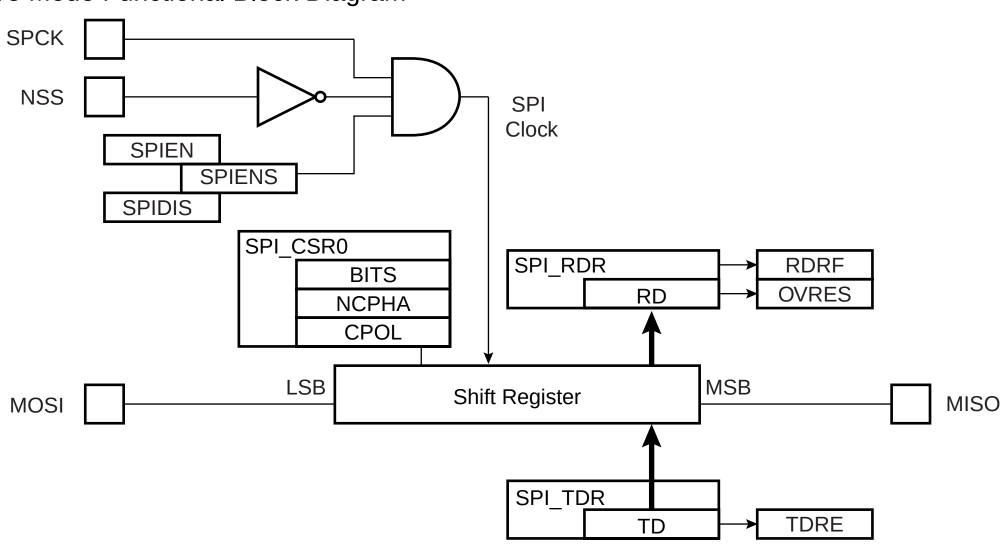
When a transfer starts, the data shifted out is the data present in the Shift Register. If no data has been written in the Transmit Data Register (SPI_TDR), the last data received is transferred. If no data has been received since the last reset, all bits are transmitted low, as the Shift Register resets at 0.

When a first data is written in SPI_TDR, it is transferred immediately in the Shift Register and the TDRE bit rises. If new data is written, it remains in SPI_TDR until a transfer occurs, i.e. NSS falls and there is a valid clock on the SPCK pin. When the transfer occurs, the last data written in SPI_TDR is transferred in the Shift Register and the TDRE bit rises. This enables frequent updates of critical variables with single transfers.

Then, a new data is loaded in the Shift Register from the Transmit Data Register. In case no character is ready to be transmitted, i.e. no character has been written in SPI_TDR since the last load from SPI_TDR to the Shift Register, the Shift Register is not modified and the last received character is retransmitted.

Figure 24-9 shows a block diagram of the SPI when operating in Slave Mode.

Figure 24-9. Slave Mode Functional Block Diagram



24.7 Serial Peripheral Interface (SPI) User Interface

Table 24-3. SPI Register Mapping

Offset	Register	Register Name	Access	Reset
0x00	Control Register	SPI_CR	Write-only	---
0x04	Mode Register	SPI_MR	Read/Write	0x0
0x08	Receive Data Register	SPI_RDR	Read-only	0x0
0x0C	Transmit Data Register	SPI_TDR	Write-only	---
0x10	Status Register	SPI_SR	Read-only	0x000000F0
0x14	Interrupt Enable Register	SPI_IER	Write-only	---
0x18	Interrupt Disable Register	SPI_IDR	Write-only	---
0x1C	Interrupt Mask Register	SPI_IMR	Read-only	0x0
0x20 - 0x2C	Reserved			
0x30	Chip Select Register 0	SPI_CSR0	Read/Write	0x0
0x34	Chip Select Register 1	SPI_CSR1	Read/Write	0x0
0x38	Chip Select Register 2	SPI_CSR2	Read/Write	0x0
0x3C	Chip Select Register 3	SPI_CSR3	Read/Write	0x0
0x004C - 0x00F8	Reserved	–	–	–
0x004C - 0x00FC	Reserved	–	–	–
0x100 - 0x124	Reserved for the PDC			

24.7.1 SPI Control Register

Name: SPI_CR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
SWRST	–	–	–	–	–	SPIDIS	SPIEN

- **SPIEN: SPI Enable**

0 = No effect.

1 = Enables the SPI to transfer and receive data.

- **SPIDIS: SPI Disable**

0 = No effect.

1 = Disables the SPI.

As soon as SPIDIS is set, SPI finishes its transfer.

All pins are set in input mode and no data is received or transmitted.

If a transfer is in progress, the transfer is finished before the SPI is disabled.

If both SPIEN and SPIDIS are equal to one when the control register is written, the SPI is disabled.

- **SWRST: SPI Software Reset**

0 = No effect.

1 = Reset the SPI. A software-triggered hardware reset of the SPI interface is performed.

The SPI is in slave mode after software reset.

PDC channels are not affected by software reset.

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

24.7.2 SPI Mode Register

Name: SPI_MR

Access Type: Read/Write

31	30	29	28	27	26	25	24
DLYBCS							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	PCS			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	–	–	–	–	–
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
LLB	–	–	MODFDIS	FDIV	PCSDEC	PS	MSTR

- **MSTR: Master/Slave Mode**

0 = SPI is in Slave mode.

1 = SPI is in Master mode.

- **PS: Peripheral Select**

0 = Fixed Peripheral Select.

1 = Variable Peripheral Select.

- **PCSDEC: Chip Select Decode**

0 = The chip selects are directly connected to a peripheral device.

1 = The four chip select lines are connected to a 4- to 16-bit decoder.

When PCSDEC equals one, up to 15 Chip Select signals can be generated with the four lines using an external 4- to 16-bit decoder. The Chip Select Registers define the characteristics of the 15 chip selects according to the following rules:

SPI_CSR0 defines peripheral chip select signals 0 to 3.

SPI_CSR1 defines peripheral chip select signals 4 to 7.

SPI_CSR2 defines peripheral chip select signals 8 to 11.

SPI_CSR3 defines peripheral chip select signals 12 to 14.

- **FDIV: 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. (MISO is internally connected on MOSI.)

- **PCS: Peripheral Chip Select**

This field is only used if Fixed Peripheral Select is active (PS = 0).

If PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don't care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS.

- **DLYBCS: Delay Between Chip Selects**

This field defines the delay from NPCS inactive to the activation of another NPCS. The DLYBCS time guarantees non-overlapping chip selects and solves bus contentions in case of peripherals having long data float times.

If DLYBCS is less than or equal to six, six MCK periods (or 6*N MCK periods if FDIV is set) will be inserted by default.

Otherwise, the following equation determines the delay:

If FDIV is 0:

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

If FDIV is 1:

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

24.7.3 SPI Receive Data Register

Name: SPI_RDR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
RD							

7	6	5	4	3	2	1	0
RD							

- **RD: Receive Data**

Data received by the SPI Interface is stored in this register right-justified. Unused bits read zero.

- **PCS: Peripheral Chip Select**

In Master Mode only, these bits indicate the value on the NPCS pins at the end of a transfer. Otherwise, these bits read zero.

24.7.4 SPI Transmit Data Register

Name: SPI_TDR

Access Type: Write-only

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

- **TD: Transmit Data**

Data to be transmitted by the SPI Interface is stored in this register. Information to be transmitted must be written to the transmit data register in a right-justified format.

PCS: Peripheral Chip Select

This field is only used if Variable Peripheral Select is active (PS = 1).

If PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don't care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS

- **LASTXFER: Last Transfer**

0 = No effect.

1 = The current NPCS will be deasserted after the character written in TD has been transferred. When CSAAT is set, this allows to close the communication with the current serial peripheral by raising the corresponding NPCS line as soon as TD transfer has completed.

This field is only used if Variable Peripheral Select is active (PS = 1).

24.7.5 SPI Status Register

Name: SPI_SR

Access Type: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	–	–	–	SPIENS
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	–	–	–	TXEMPTY	NSSR
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full**

0 = No data has been received since the last read of SPI_RDR

1 = Data has been received and the received data has been transferred from the serializer to SPI_RDR since the last read of SPI_RDR.

- **TDRE: Transmit Data Register Empty**

0 = Data has been written to SPI_TDR and not yet transferred to the serializer.

1 = The last data written in the Transmit Data Register has been transferred to the serializer.

TDRE equals zero when the SPI is disabled or at reset. The SPI enable command sets this bit to one.

- **MODF: Mode Fault Error**

0 = No Mode Fault has been detected since the last read of SPI_SR.

1 = A Mode Fault occurred since the last read of the SPI_SR.

- **OVRES: Overrun Error Status**

0 = No overrun has been detected since the last read of SPI_SR.

1 = An overrun has occurred since the last read of SPI_SR.

An overrun occurs when SPI_RDR is loaded at least twice from the serializer since the last read of the SPI_RDR.

- **ENDRX: End of RX buffer**

0 = The Receive Counter Register has not reached 0 since the last write in SPI_RCR[(1)] or SPI_RNCR[(1)].

1 = The Receive Counter Register has reached 0 since the last write in SPI_RCR[(1)] or SPI_RNCR[(1)].

- **ENDTX: End of TX buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in SPI_TCR[(1)] or SPI_TNCR[(1)].

1 = The Transmit Counter Register has reached 0 since the last write in SPI_TCR[(1)] or SPI_TNCR[(1)].

- **RXBUFF: RX Buffer Full**

0 = SPI_RCR[(1)] or SPI_RNCR[(1)] has a value other than 0.

1 = Both SPI_RCR[(1)] and SPI_RNCR[(1)] have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = SPI_TCR[(1)] or SPI_TNCR[(1)] has a value other than 0.

1 = Both SPI_TCR[(1)] and SPI_TNCR[(1)] have a value of 0.

- **NSSR: NSS Rising**

0 = No rising edge detected on NSS pin since last read.

1 = A rising edge occurred on NSS pin since last read.

- **TXEMPTY: Transmission Registers Empty**

0 = As soon as data is written in SPI_TDR.

1 = SPI_TDR and internal shifter are empty. If a transfer delay has been defined, TXEMPTY is set after the completion of such delay.

- **SPIENS: SPI Enable Status**

0 = SPI is disabled.

1 = SPI is enabled.

Note: 1. SPI_RCR, SPI_RNCR, SPI_TCR, SPI_TNCR are physically located in the PDC.

24.7.6 SPI Interrupt Enable Register

Name: SPI_IER

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	–	–	–	–
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	–	–	–	TXEMPTY	NSSR
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full Interrupt Enable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Enable**
- **MODF: Mode Fault Error Interrupt Enable**
- **OVRES: Overrun Error Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**
- **TXEMPTY: Transmission Registers Empty Enable**
- **NSSR: NSS Rising Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

24.7.7 SPI Interrupt Disable Register

Name: SPI_IDR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	TXEMPTY	NSSR

7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full Interrupt Disable**
- **TDRE: SPI Transmit Data Register Empty Interrupt Disable**
- **MODF: Mode Fault Error Interrupt Disable**
- **OVRES: Overrun Error Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**
- **TXEMPTY: Transmission Registers Empty Disable**
- **NSSR: NSS Rising Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

24.7.8 SPI Interrupt Mask Register

Name: SPI_IMR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	TXEMPTY	NSSR

7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

- **RDRF: Receive Data Register Full Interrupt Mask**
- **TDRE: SPI Transmit Data Register Empty Interrupt Mask**
- **MODF: Mode Fault Error Interrupt Mask**
- **OVRES: Overrun Error Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**
- **TXEMPTY: Transmission Registers Empty Mask**
- **NSSR: NSS Rising Interrupt Mask**

0 = The corresponding interrupt is not enabled.

1 = The corresponding interrupt is enabled.

24.7.9 SPI Chip Select Register

Name: SPI_CSR0... SPI_CSR3

Access Type: Read/Write

31	30	29	28	27	26	25	24
DLYBCT							

23	22	21	20	19	18	17	16
DLYBS							

15	14	13	12	11	10	9	8
SCBR							

7	6	5	4	3	2	1	0
BITS				CSAAT	–	NCPHA	CPOL

- **CPOL: Clock Polarity**

0 = The inactive state value of SPCK is logic level zero.

1 = The inactive state value of SPCK is logic level one.

CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the required clock/data relationship between master and slave devices.

- **NCPHA: Clock Phase**

0 = Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.

1 = Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.

NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is used with CPOL to produce the required clock/data relationship between master and slave devices.

- **CSAAT: Chip Select Active After Transfer**

0 = The Peripheral Chip Select Line rises as soon as the last transfer is achieved.

1 = The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is requested on a different chip select.

- **BITS: Bits Per Transfer**

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

BITS	Bits Per Transfer
0000	8
0001	9
0010	10
0011	11
0100	12
0101	13
0110	14
0111	15
1000	16
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	Reserved
1111	Reserved

• **SCBR: Serial Clock Baud Rate**

In Master Mode, the SPI Interface uses a modulus counter to derive the SPCK baud rate from the Master Clock MCK. The Baud rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK baud rate:

If FDIV is 0:

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

If FDIV is 1:

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

Note: N = 32

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.

At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.

• **DLYBS: Delay Before SPCK**

This field defines the delay from NPCS valid to the first valid SPCK transition.

When DLYBS equals zero, the NPCS valid to SPCK transition is 1/2 the SPCK clock period.

Otherwise, the following equations determine the delay:

If FDIV is 0:

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

If FDIV is 1:

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

Note: N = 32

- **DLYBCT: Delay Between Consecutive Transfers**

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

When DLYBCT equals zero, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the character transfers.

Otherwise, the following equation determines the delay:

If FDIV is 0:

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

If FDIV is 1:

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

Note: N = 32

25. Two-wire Interface (TWI)

25.1 Description

The Atmel Two-wire Interface (TWI) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 Kbits per second, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus Serial EEPROM and I²C compatible device such as Real Time Clock (RTC), Dot Matrix/Graphic LCD Controllers and Temperature Sensor, to name but a few. The TWI is programmable as a master or a slave with sequential or single-byte access. Multiple master capability is supported. Arbitration of the bus is performed internally and puts the TWI in slave mode automatically if the bus arbitration is lost.

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

Below, Table 25-1 lists the compatibility level of the Atmel Two-wire Interface in Master Mode and a full I2C compatible device.

Table 25-1. Atmel TWI compatibility with i2C Standard

I2C Standard	Atmel TWI
Standard Mode Speed (100 KHz)	Supported
Fast Mode Speed (400 KHz)	Supported
7 or 10 bits Slave Addressing	Supported
START BYTE [(1)]	Not Supported
Repeated Start (Sr) Condition	Supported
ACK and NACK Management	Supported
Slope control and input filtering (Fast mode)	Not Supported
Clock stretching	Supported

1. START + b000000001 + Ack + Sr

25.2 List of Abbreviations

Table 25-2. Abbreviations

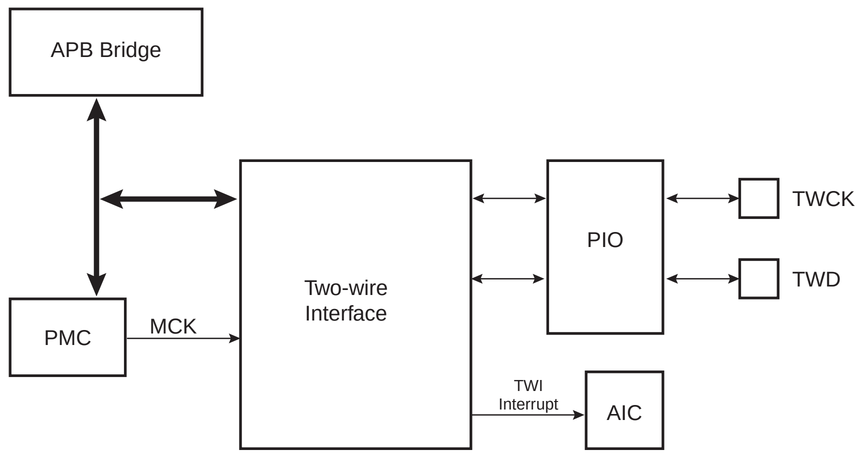
Abbreviation	Description
TWI	Two-wire Interface
A	Acknowledge
NA	Non Acknowledge
P	Stop
S	Start
Sr	Repeated Start
SADR	Slave Address

Table 25-2. Abbreviations

Abbreviation	Description
ADR	Any address except SADR
R	Read
W	Write

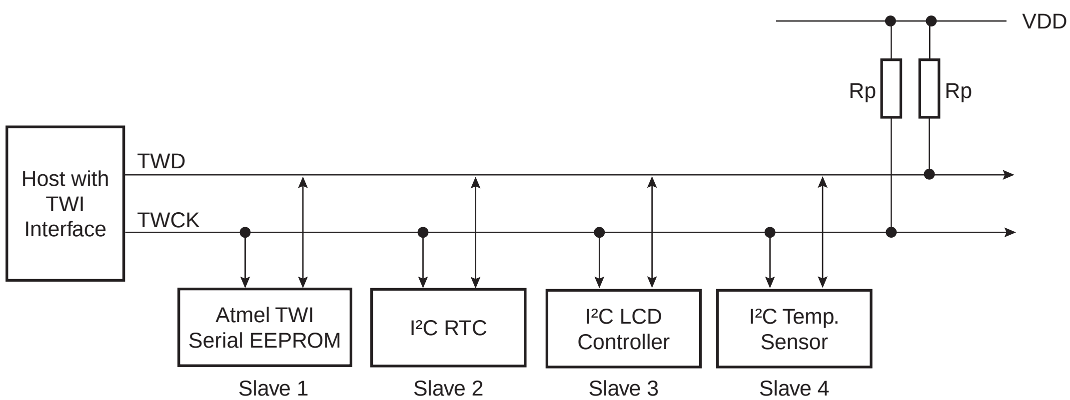
25.3 Block Diagram

Figure 25-1. Block Diagram



25.4 Application Block Diagram

Figure 25-2. Application Block Diagram



Rp: Pull up value as given by the I²C Standard

25.4.1 I/O Lines Description

Table 25-3. I/O Lines Description

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

25.5 Product Dependencies

25.5.1 I/O Lines

Both TWD and TWCK are bidirectional lines, connected to a positive supply voltage via a current source or pull-up resistor (see Figure 25-2). When the bus is free, both lines are high. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function.

TWD and TWCK pins may be multiplexed with PIO lines. To enable the TWI, the programmer must perform the following step:

- Program the PIO controller to dedicate TWD and TWCK as peripheral lines.

The user must not program TWD and TWCK as open-drain. It is already done by the hardware.

25.5.2 Power Management

- Enable the peripheral clock.

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

25.5.3 Interrupt

The TWI interface has an interrupt line connected to the Advanced Interrupt Controller (AIC). In order to handle interrupts, the AIC must be programmed before configuring the TWI.

25.6 Functional Description

25.6.1 Transfer Format

The data put on the TWD line must be 8 bits long. Data is transferred MSB first; each byte must be followed by an acknowledgement. The number of bytes per transfer is unlimited (see Figure 25-4).

Each transfer begins with a START condition and terminates with a STOP condition (see Figure 25-3).

- A high-to-low transition on the TWD line while TWCK is high defines the START condition.
- A low-to-high transition on the TWD line while TWCK is high defines a STOP condition.

Figure 25-3. START and STOP Conditions

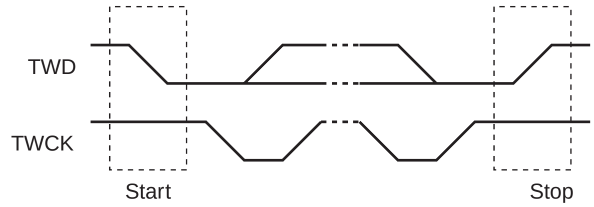
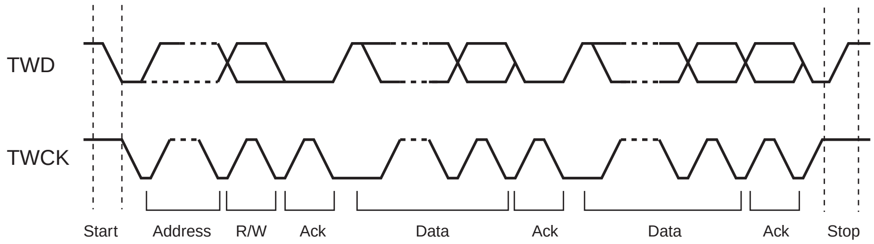


Figure 25-4. Transfer Format



25.6.2 Modes of Operation

The TWI has six modes of operations:

- Master transmitter mode
- Master receiver mode
- Multi-master transmitter mode
- Multi-master receiver mode
- Slave transmitter mode
- Slave receiver mode

These modes are described in the following chapters.

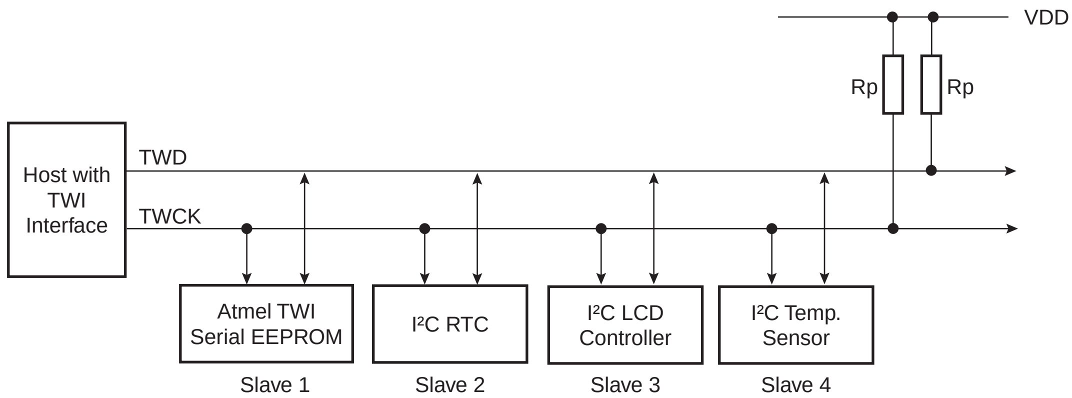
25.7 Master Mode

25.7.1 Definition

The Master is the device that starts a transfer, generates a clock and stops it.

25.7.2 Application Block Diagram

Figure 25-5. Master Mode Typical Application Block Diagram



25.7.3 Programming Master Mode

The following registers have to be programmed before entering Master mode:

1. DADR (+ IADRSZ + IADR if a 10 bit device is addressed): The device address is used to access slave devices in read or write mode.
2. CKDIV + CHDIV + CLDIV: Clock Waveform.
3. SVDIS: Disable the slave mode.
4. MSEN: Enable the master mode.

25.7.4 Master Transmitter Mode

After the master initiates a Start condition when writing into the Transmit Holding Register, TWI_THR, it sends a 7-bit slave address, configured in the Master Mode register (DADR in TWI_MMR), to notify the slave device. The bit following the slave address indicates the transfer direction, 0 in this case (MREAD = 0 in TWI_MMR).

The TWI transfers require the slave to acknowledge each received byte. During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the Not Acknowledge bit (**NACK)** in the status register if the slave does not acknowledge the byte. As with the other status bits, an interrupt can be generated if enabled in the interrupt enable register (TWI_IER). If the slave acknowledges the byte, the data written in the TWI_THR, is then shifted in the internal shifter and transferred. When an acknowledge is detected, the TXRDY bit is set until a new write in the TWI_THR.

While no new data is written in the TWI_THR, the Serial Clock Line is tied low. When new data is written in the TWI_THR, the SCL is released and the data is sent. To generate a STOP event, the STOP command must be performed by writing in the STOP field of TWI_CR.

After a Master Write transfer, the Serial Clock line is stretched (tied low) while no new data is written in the TWI_THR or until a STOP command is performed.

See Figure 25-6, Figure 25-7, and Figure 25-8.

Figure 25-6. Master Write with One Data Byte

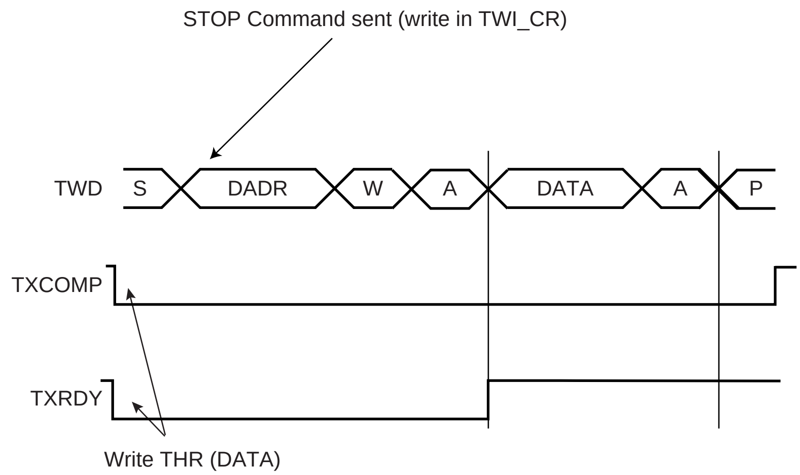


Figure 25-7. Master Write with Multiple Data Bytes

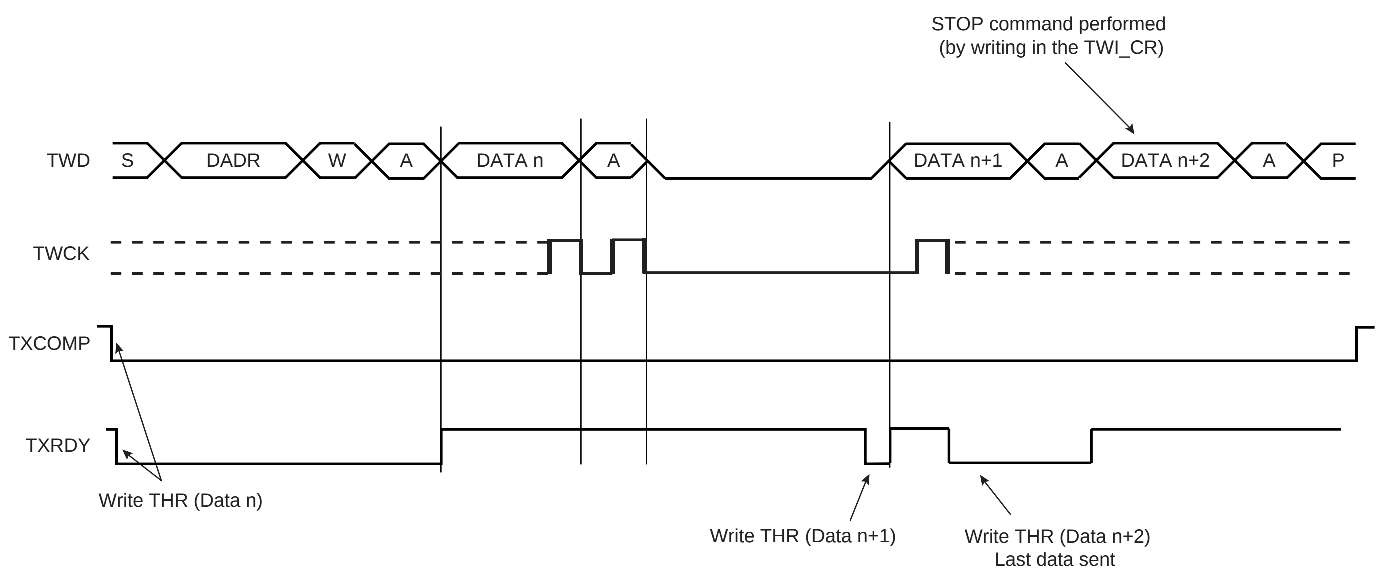
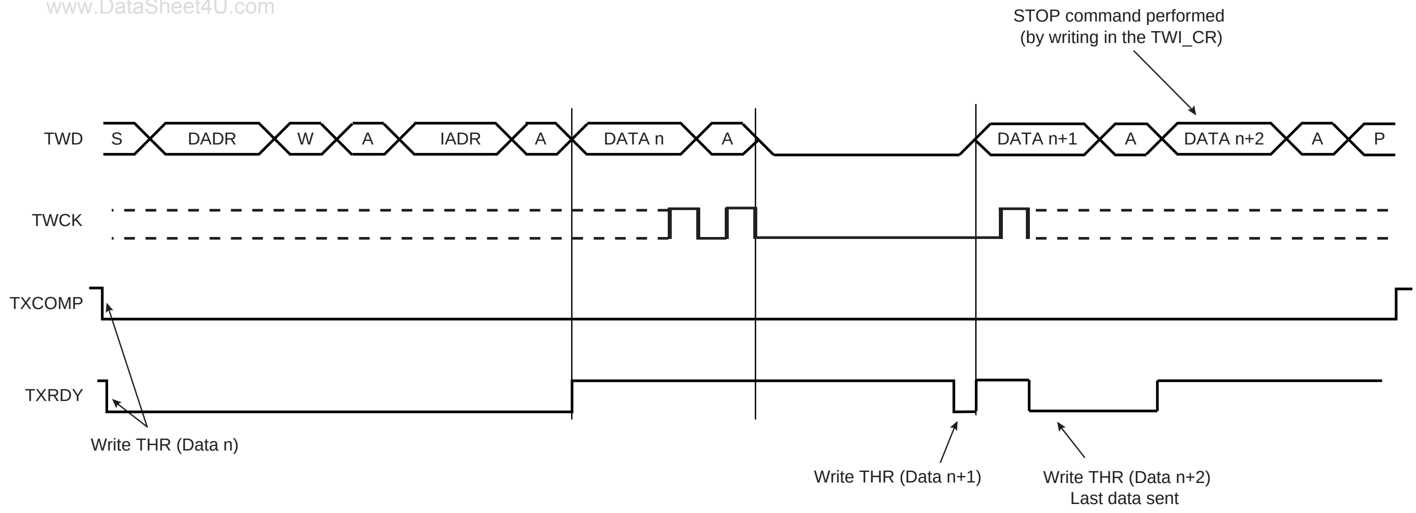


Figure 25-8. Master Write with One Byte Internal Address and Multiple Data Bytes



TXRDY is used as Transmit Ready for the PDC transmit channel.

25.7.5 Master Receiver Mode

The read sequence begins by setting the START bit. After the start condition has been sent, the master sends a 7-bit slave address to notify the slave device. The bit following the slave address indicates the transfer direction, 1 in this case (MREAD = 1 in TWI_MMR). During the acknowledge clock pulse (9th pulse), the master releases the data line (HIGH), enabling the slave to pull it down in order to generate the acknowledge. The master polls the data line during this clock pulse and sets the **NACK** bit in the status register if the slave does not acknowledge the byte.

If an acknowledge is received, the master is then ready to receive data from the slave. After data has been received, the master sends an acknowledge condition to notify the slave that the data has been received except for the last data, after the stop condition. See Figure 25-9. When the RXRDY bit is set in the status register, a character has been received in the receive-holding register (TWI_RHR). The RXRDY bit is reset when reading the TWI_RHR.

When a single data byte read is performed, with or without internal address **(IADR**), the START and STOP bits must be set at the same time. See Figure 25-9. When a multiple data byte read is performed, with or without internal address **(IADR**), the STOP bit must be set after the next-to-last data received. See Figure 25-10. For Internal Address usage see Section 25.7.6.

Figure 25-9. Master Read with One Data Byte

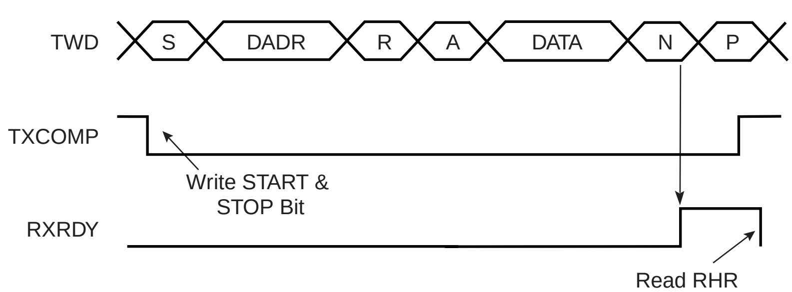
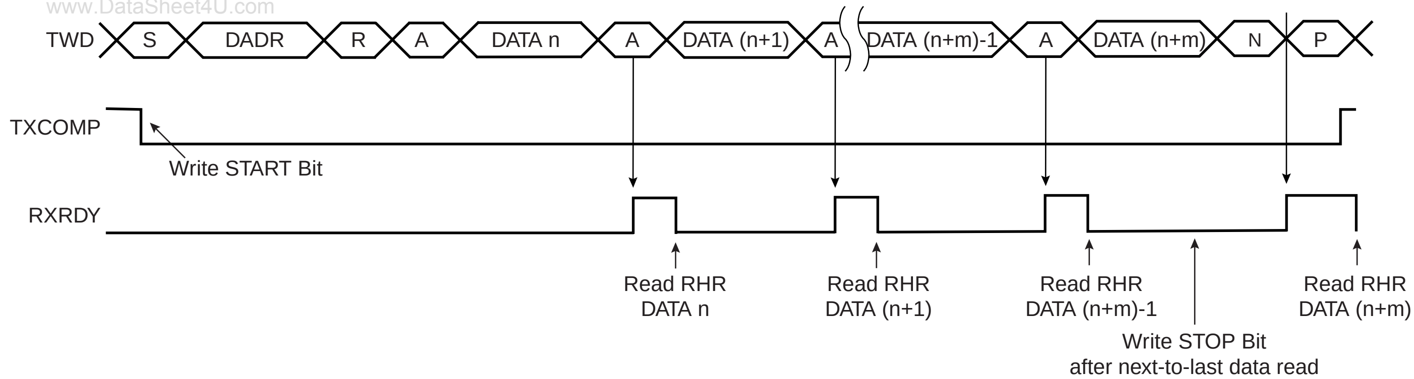


Figure 25-10. Master Read with Multiple Data Bytes



RXRDY is used as Receive Ready for the PDC receive channel.

25.7.6 Internal Address

The TWI interface can perform various transfer formats: Transfers with 7-bit slave address devices and 10-bit slave address devices.

25.7.6.1 7-bit Slave Addressing

When Addressing 7-bit slave devices, the internal address bytes are used to perform random address (read or write) accesses to reach one or more data bytes, within a memory page location in a serial memory, for example. When performing read operations with an internal address, the TWI performs a write operation to set the internal address into the slave device, and then switch to Master Receiver mode. Note that the second start condition (after sending the IADR) is sometimes called "repeated start" (Sr) in I2C fully-compatible devices. See Figure 25-12. See Figure 25-11 and Figure 25-13 for Master Write operation with internal address.

The three internal address bytes are configurable through the Master Mode register (TWI_MMR).

If the slave device supports only a 7-bit address, i.e. no internal address, **IADRSZ** must be set to 0.

In the figures below the following abbreviations are used:

- **S** Start
- **Sr** Repeated Start
- **P** Stop
- **W** Write
- **R** Read
- **A** Acknowledge
- **N** Not Acknowledge
- **DADR** Device Address
- **IADR** Internal Address

Figure 25-11. Master Write with One, Two or Three Bytes Internal Address and One Data Byte

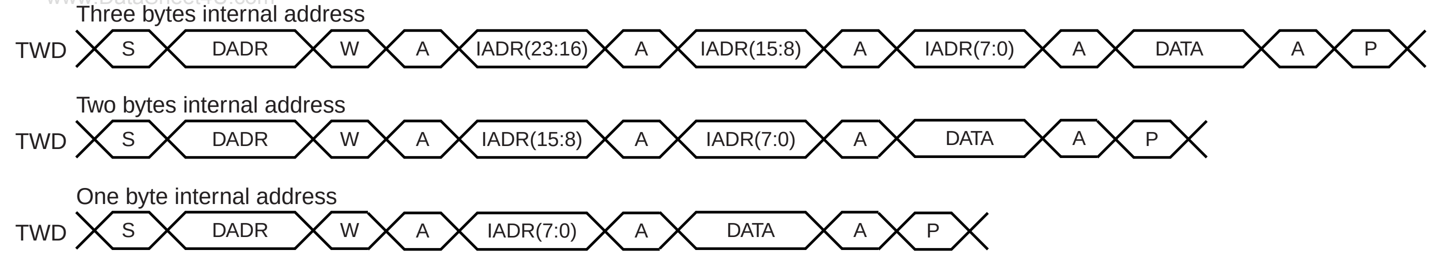
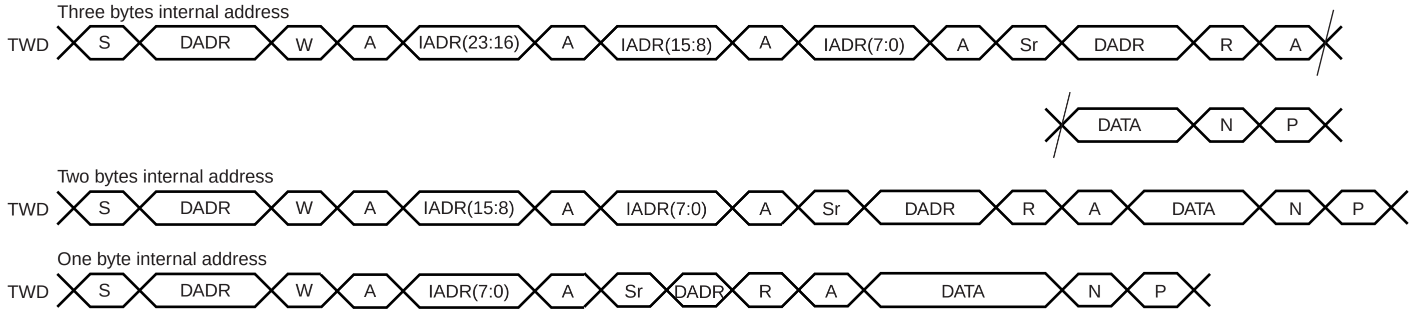


Figure 25-12. Master Read with One, Two or Three Bytes Internal Address and One Data Byte



25.7.6.2 *10-bit Slave Addressing*

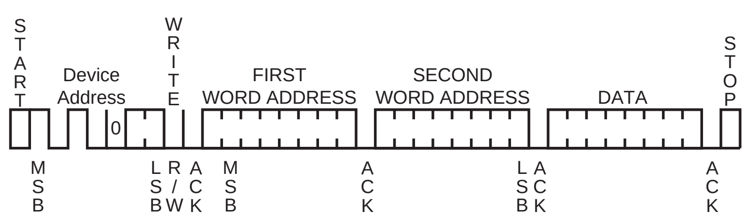
For a slave address higher than 7 bits, the user must configure the address size **(IADRSZ**) and set the other slave address bits in the internal address register (TWI_IADR). The two remaining Internal address bytes, IADR[15:8] and IADR[23:16] can be used the same as in 7-bit Slave Addressing.

__**Example:** Address a 10-bit device__ (10-bit device address is b1 b2 b3 b4 b5 b6 b7 b8 b9 b10)

1. Program IADRSZ = 1,
2. Program DADR with 1 1 1 1 0 b1 b2 (b1 is the MSB of the 10-bit address, b2, etc.)
3. Program TWI_IADR with b3 b4 b5 b6 b7 b8 b9 b10 (b10 is the LSB of the 10-bit address)

Figure 25-13 below shows a byte write to an Atmel AT24LC512 EEPROM. This demonstrates the use of internal addresses to access the device.

Figure 25-13. Internal Address Usage



25.7.7 Using the Peripheral DMA Controller (PDC)

The use of the PDC significantly reduces the CPU load.

To assure correct implementation, respect the following programming sequences:

25.7.7.1 Data Transmit with the PDC

1. Initialize the transmit PDC (memory pointers, size, etc.).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Start the transfer by setting the PDC TXTEN bit.
4. Wait for the PDC end TX flag.
5. Disable the PDC by setting the PDC TXDIS bit.

25.7.7.2 Data Receive with the PDC

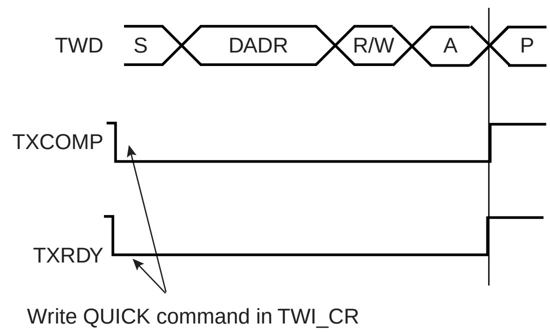
1. Initialize the receive PDC (memory pointers, size - 1, etc.).
2. Configure the master mode (DADR, CKDIV, etc.).
3. Start the transfer by setting the PDC RXTEN bit.
4. Wait for the PDC end RX flag.
5. Disable the PDC by setting the PDC RXDIS bit.

25.7.8 SMBUS Quick Command (Master Mode Only)

The TWI interface can perform a Quick Command:

1. Configure the master mode (DADR, CKDIV, etc.).
2. Write the MREAD bit in the TWI_MMR register at the value of the one-bit command to be sent.
3. Start the transfer by setting the QUICK bit in the TWI_CR.

Figure 25-14. SMBUS Quick Command



25.7.9 Read-write Flowcharts

The following flowcharts shown in Figure 25-16, Figure 25-17, Figure 25-18, Figure 25-19 and Figure 25-20 give examples for read and write operations. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI_IER) be configured first.

Figure 25-15. TWI Write Operation with Single Data Byte without Internal Address

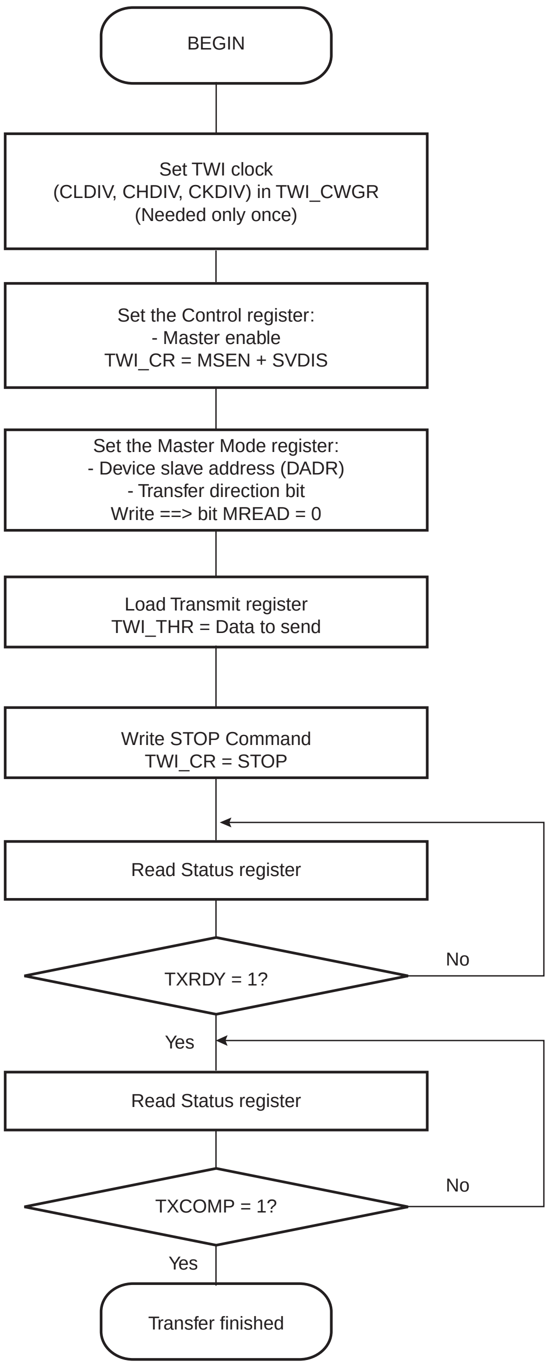


Figure 25-16. TWI Write Operation with Single Data Byte and Internal Address

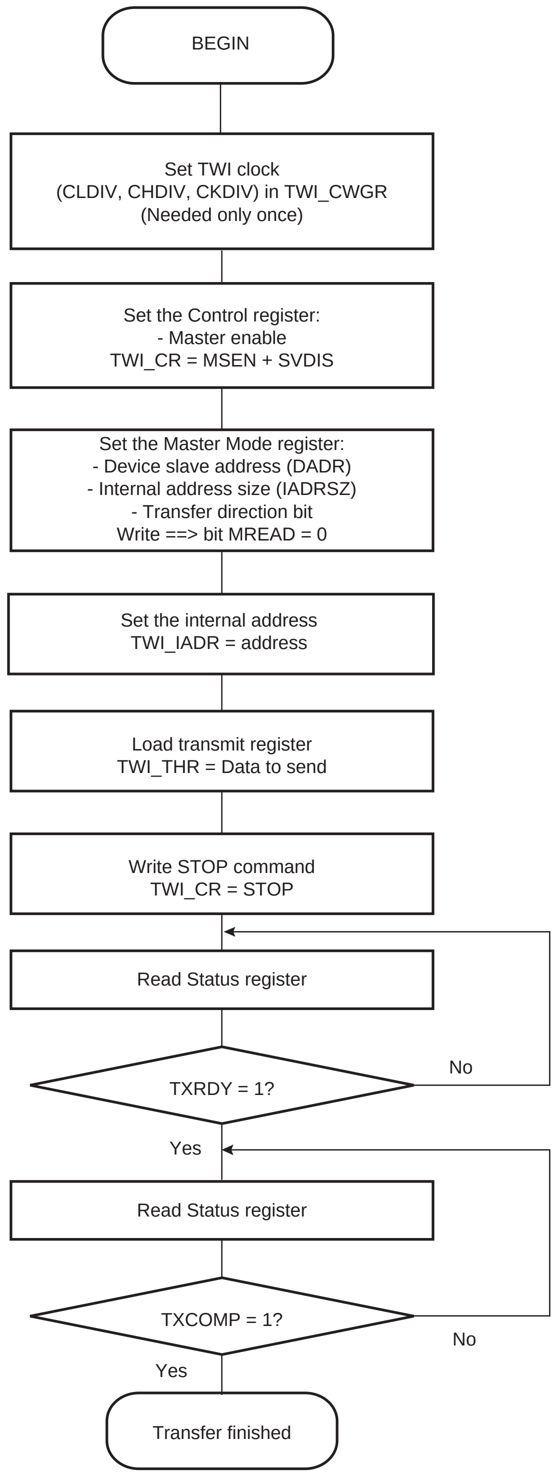


Figure 25-17. TWI Write Operation with Multiple Data Bytes with or without Internal Address

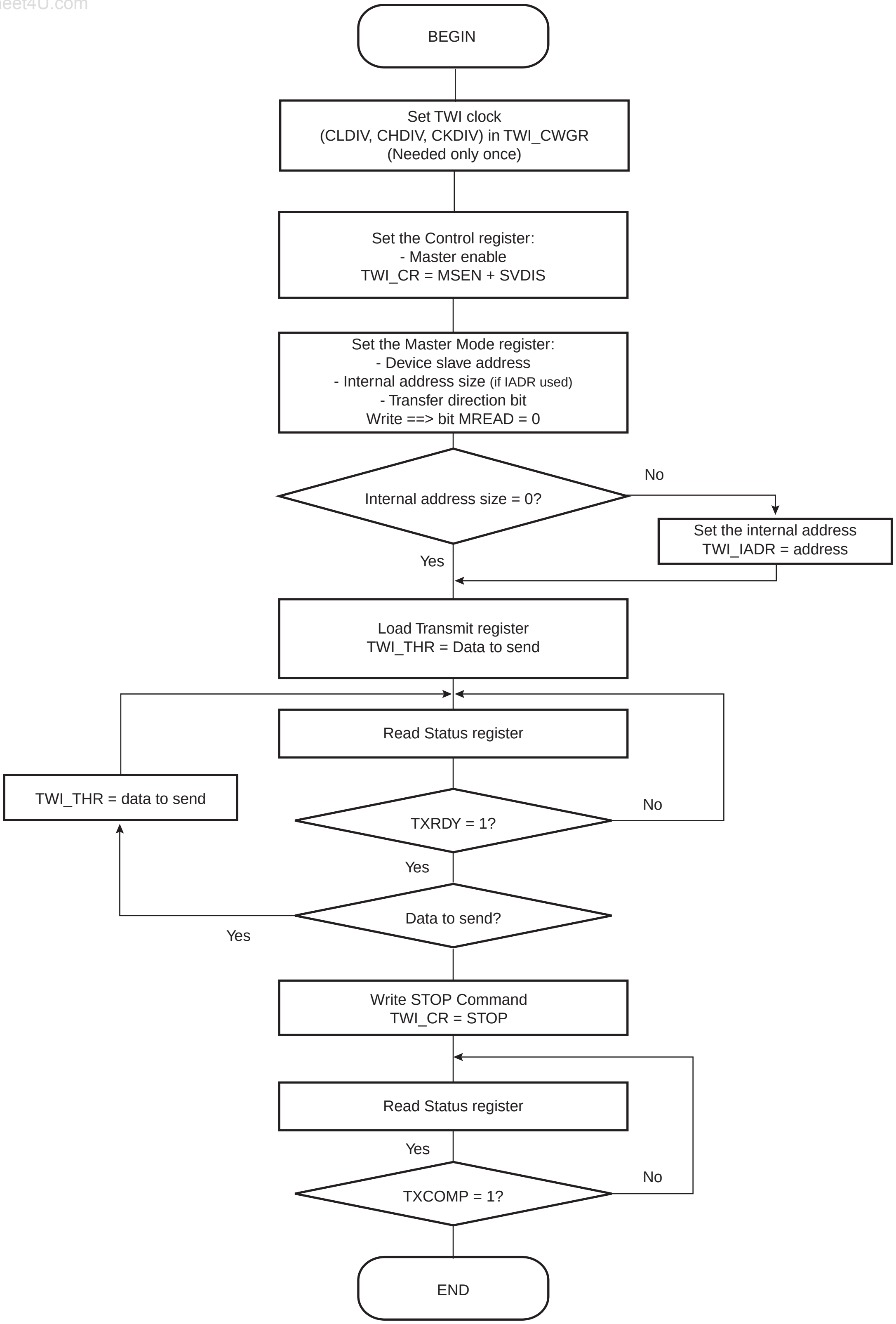


Figure 25-18. TWI Read Operation with Single Data Byte without Internal Address

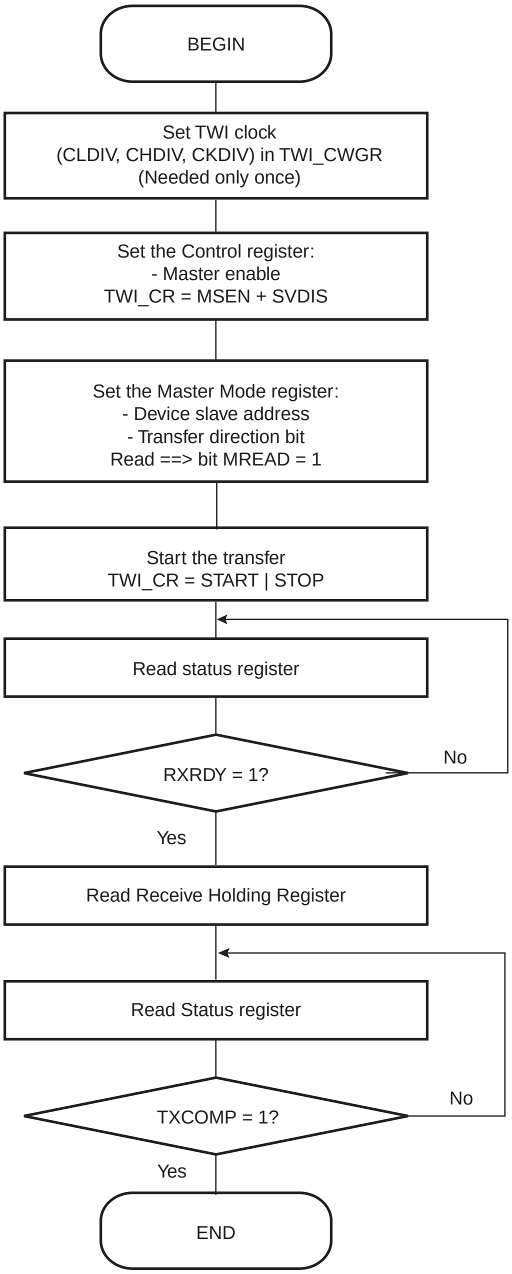


Figure 25-19. TWI Read Operation with Single Data Byte and Internal Address

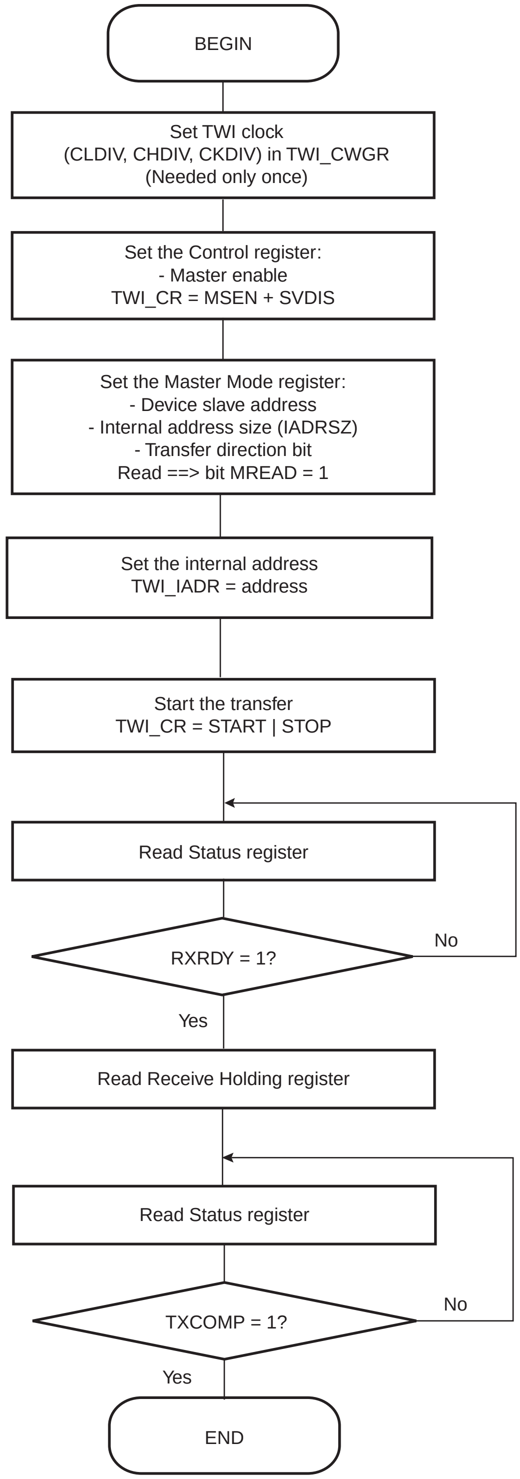
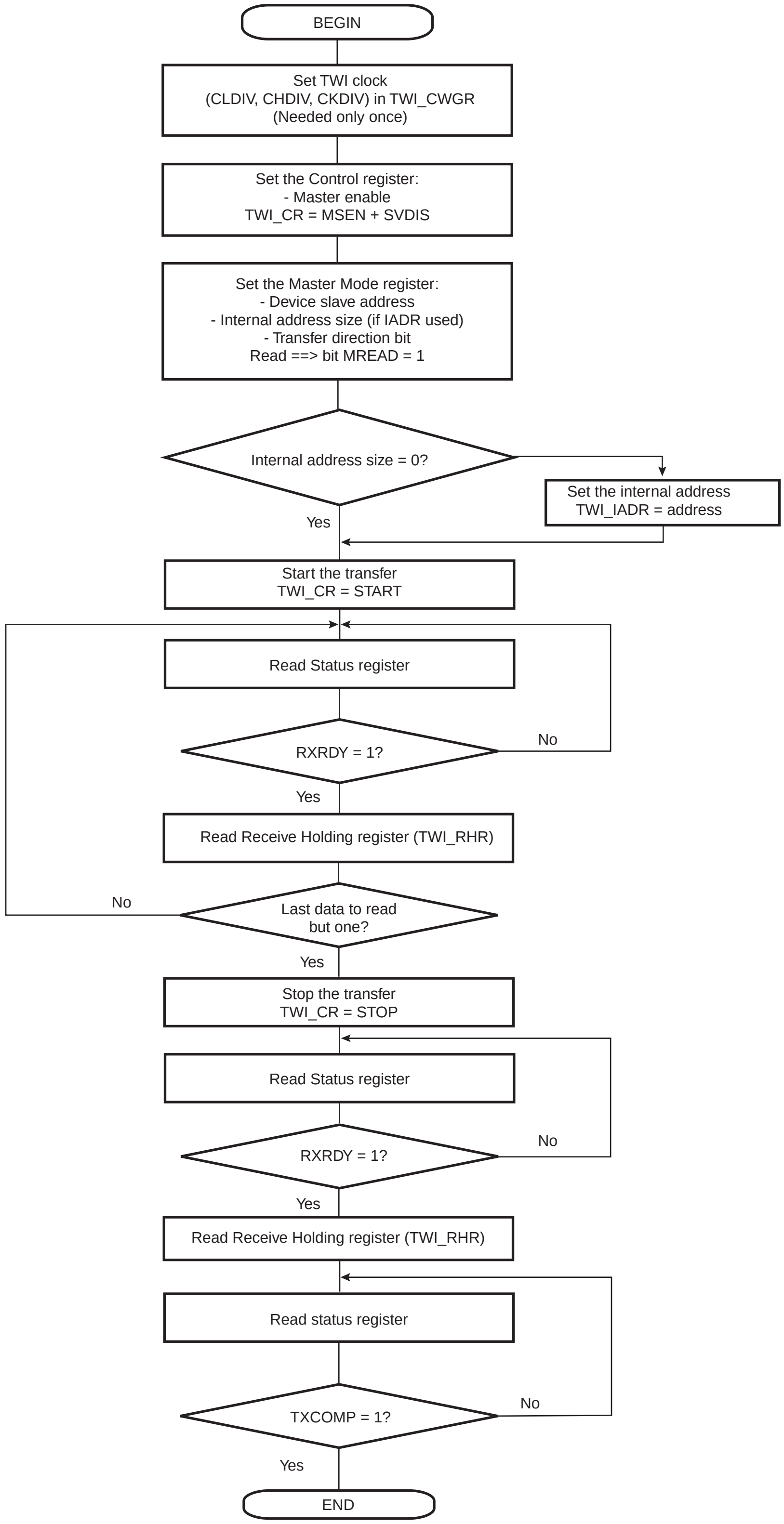


Figure 25-20. TWI Read Operation with Multiple Data Bytes with or without Internal Address



25.8 Multi-master Mode

25.8.1 Definition

More than one master may handle the bus at the same time without data corruption by using arbitration.

Arbitration starts as soon as two or more masters place information on the bus at the same time, and stops (arbitration is lost) for the master that intends to send a logical one while the other master sends a logical zero.

As soon as arbitration is lost by a master, it stops sending data and listens to the bus in order to detect a stop. When the stop is detected, the master who has lost arbitration may put its data on the bus by respecting arbitration.

Arbitration is illustrated in Figure 25-22.

25.8.2 Different Multi-master Modes

Two multi-master modes may be distinguished:

1. TWI is considered as a Master only and will never be addressed.
2. TWI may be either a Master or a Slave and may be addressed.

Note: In both Multi-master modes arbitration is supported.

25.8.2.1 TWI as Master Only

In this mode, TWI is considered as a Master only (MSEN is always at one) and must be driven like a Master with the ARBLST (ARBitration Lost) flag in addition.

If arbitration is lost (ARBLST = 1), the programmer must reinitiate the data transfer.

If the user starts a transfer (ex.: DADR + START + W + Write in THR) and if the bus is busy, the TWI automatically waits for a STOP condition on the bus to initiate the transfer (see Figure 25-21).

Note: The state of the bus (busy or free) is not indicated in the user interface.

25.8.2.2 TWI as Master or Slave

The automatic reversal from Master to Slave is not supported in case of a lost arbitration.

Then, in the case where TWI may be either a Master or a Slave, the programmer must manage the pseudo Multi-master mode described in the steps below.

1. Program TWI in Slave mode (SADR + MSDIS + SVEN) and perform Slave Access (if TWI is addressed).
2. If TWI has to be set in Master mode, wait until TXCOMP flag is at 1.
3. Program Master mode (DADR + SVDIS + MSEN) and start the transfer (ex: START + Write in THR).
4. As soon as the Master mode is enabled, TWI scans the bus in order to detect if it is busy or free. When the bus is considered as free, TWI initiates the transfer.
5. As soon as the transfer is initiated and until a STOP condition is sent, the arbitration becomes relevant and the user must monitor the ARBLST flag.
6. If the arbitration is lost (ARBLST is set to 1), the user must program the TWI in Slave mode in the case where the Master that won the arbitration wanted to access the TWI.

7. If TWI has to be set in Slave mode, wait until TXCOMP flag is at 1 and then program the Slave mode.

Note: In the case where the arbitration is lost and TWI is addressed, TWI will not acknowledge even if it is programmed in Slave mode as soon as ARBLST is set to 1. Then, the Master must repeat SADR.

Figure 25-21. Programmer Sends Data While the Bus is Busy

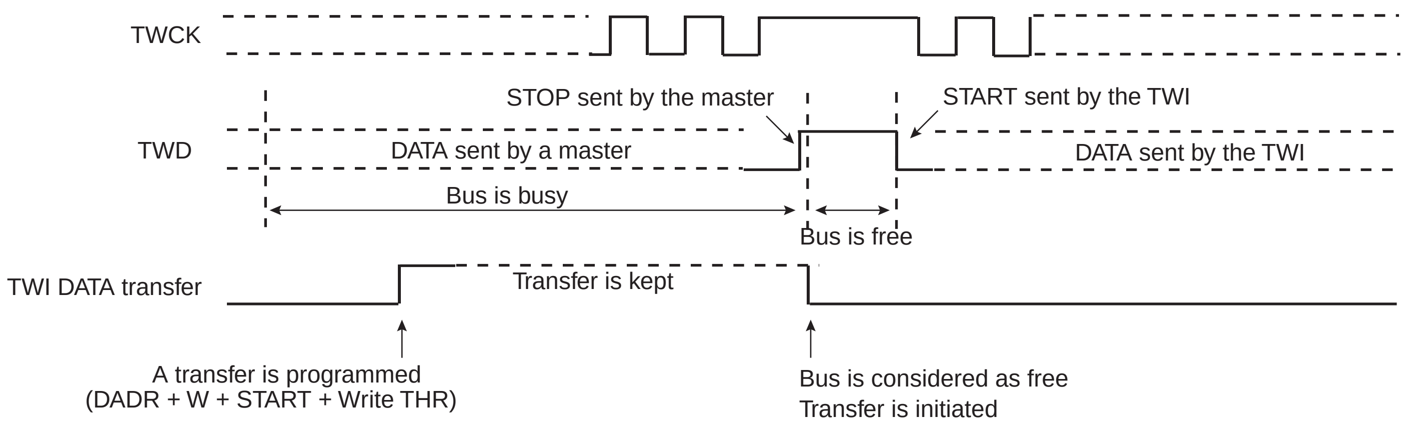
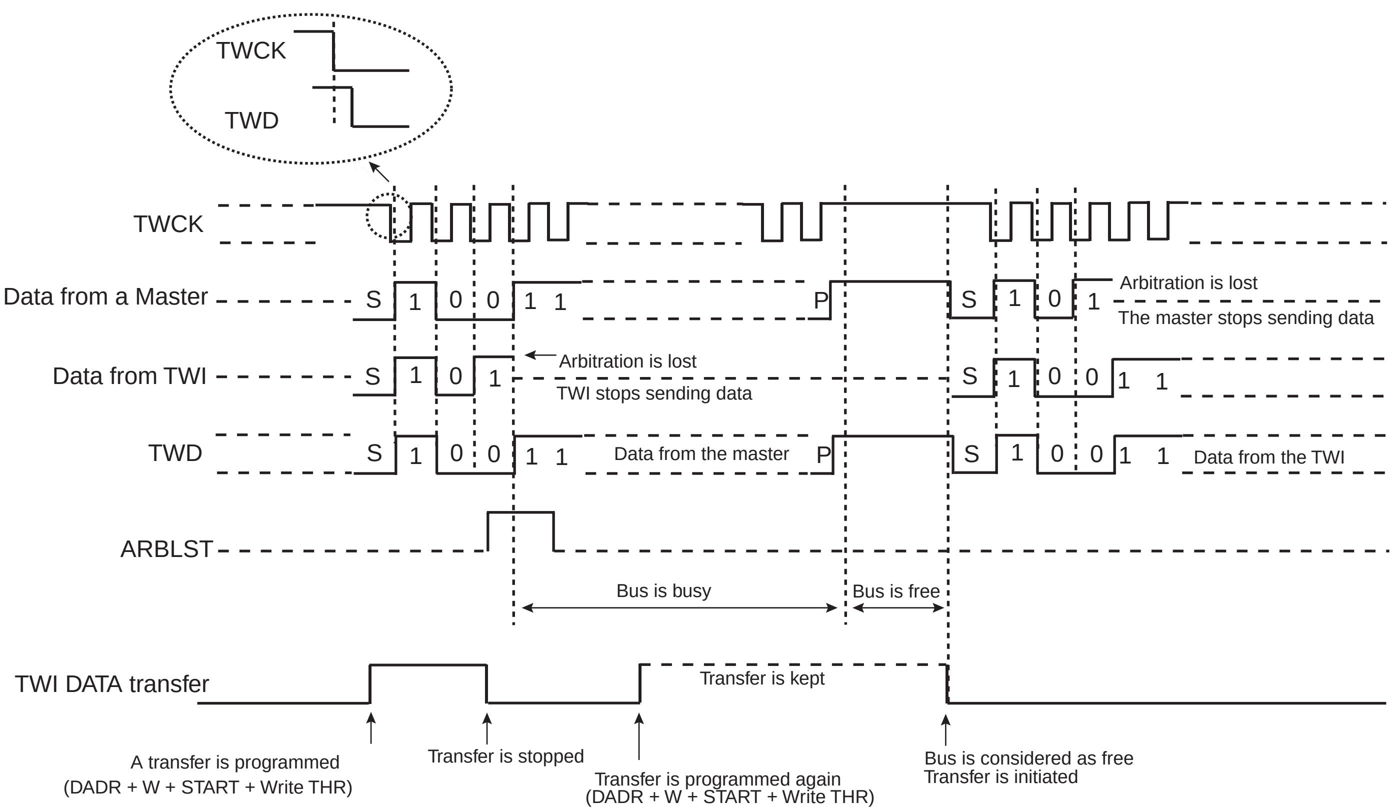
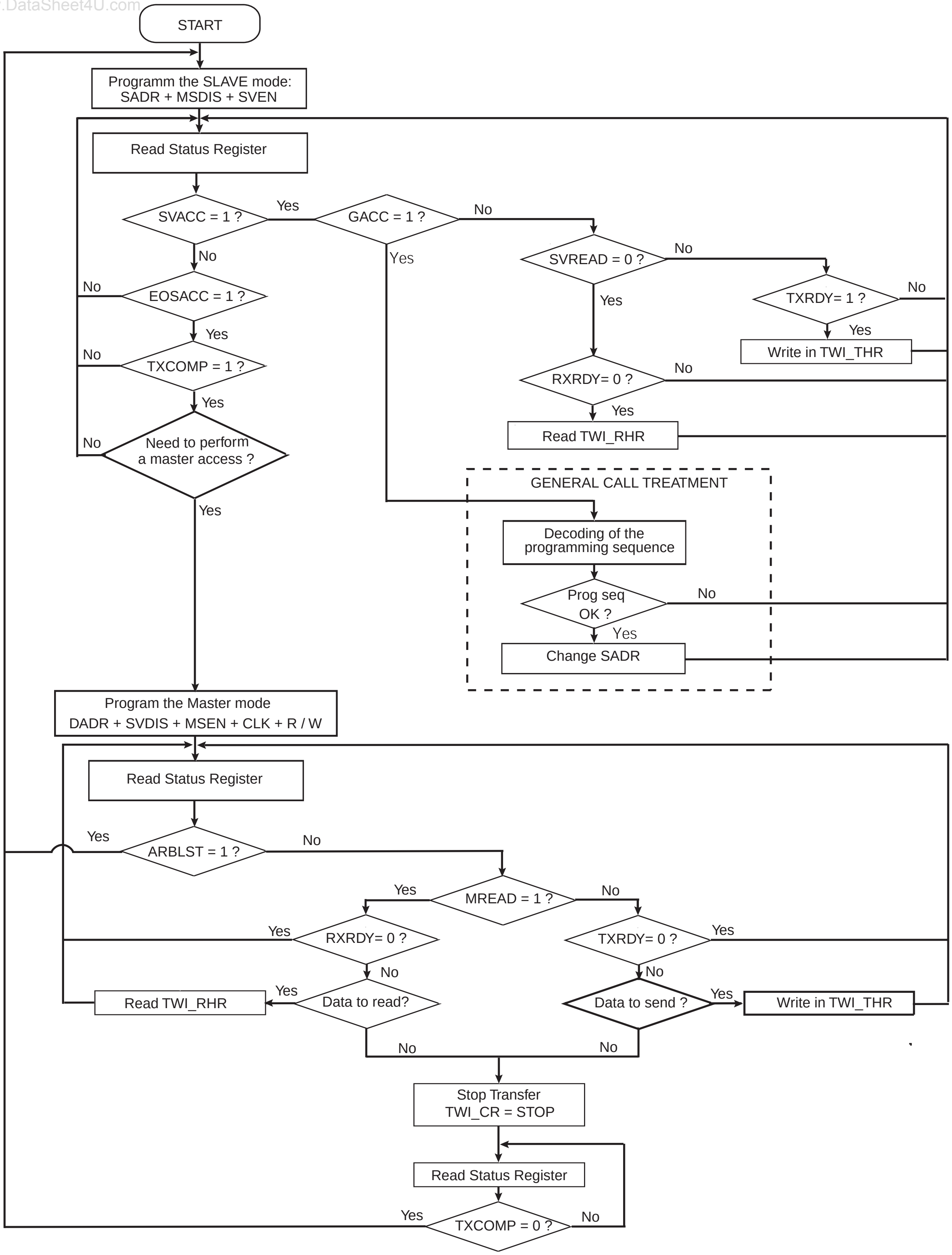


Figure 25-22. Arbitration Cases



The flowchart shown in Figure 25-23 on page 385 gives an example of read and write operations in Multi-master mode.

Figure 25-23. Multi-master Flowchart



25.9 Slave Mode

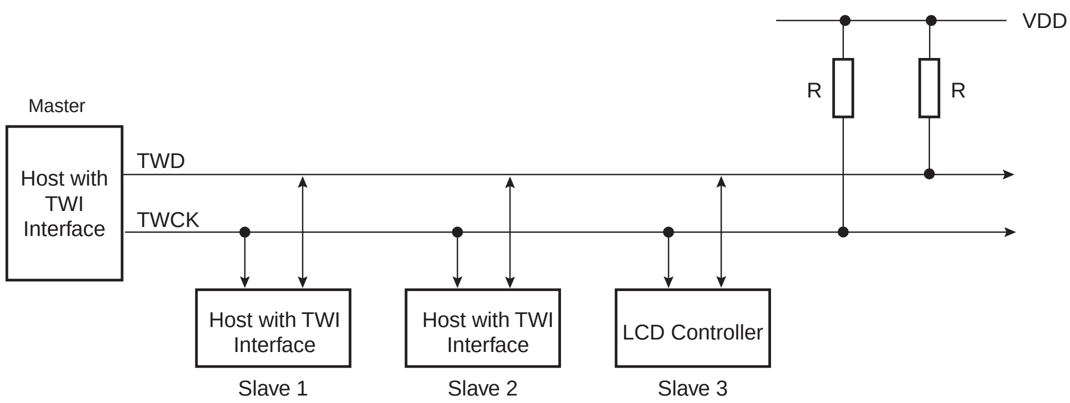
25.9.1 Definition

The Slave Mode is defined as a mode where the device receives the clock and the address from another device called the master.

In this mode, the device never initiates and never completes the transmission (START, REPEATED_START and STOP conditions are always provided by the master).

25.9.2 Application Block Diagram

Figure 25-24. Slave Mode Typical Application Block Diagram



25.9.3 Programming Slave Mode

The following fields must be programmed before entering Slave mode:

1. SADR (TWI_SMR): The slave device address is used in order to be accessed by master devices in read or write mode.
2. MSDIS (TWI_CR): Disable the master mode.
3. SVEN (TWI_CR): Enable the slave mode.

As the device receives the clock, values written in TWI_CWGR are not taken into account.

25.9.4 Receiving Data

After a Start or Repeated Start condition is detected and if the address sent by the Master matches with the Slave address programmed in the SADR (Slave ADdress) field, SVACC (Slave ACCess) flag is set and SVREAD (Slave READ) indicates the direction of the transfer.

SVACC remains high until a STOP condition or a repeated START is detected. When such a condition is detected, EOSACC (End Of Slave ACCess) flag is set.

25.9.4.1 Read Sequence

In the case of a Read sequence (SVREAD is high), TWI transfers data written in the TWI_THR (TWI Transmit Holding Register) until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the read sequence TXCOMP (Transmission Complete) flag is set and SVACC reset.

As soon as data is written in the TWI_THR, TXRDY (Transmit Holding Register Ready) flag is reset, and it is set when the shift register is empty and the sent data acknowledged or not. If the data is not acknowledged, the NACK flag is set.

Note that a STOP or a repeated START always follows a NACK.

See Figure 25-25 on page 388.

25.9.4.2 Write Sequence

In the case of a Write sequence (SVREAD is low), the RXRDY (Receive Holding Register Ready) flag is set as soon as a character has been received in the TWI_RHR (TWI Receive Holding Register). RXRDY is reset when reading the TWI_RHR.

TWI continues receiving data until a STOP condition or a REPEATED_START + an address different from SADR is detected. Note that at the end of the write sequence TXCOMP flag is set and SVACC reset.

See Figure 25-26 on page 388.

25.9.4.3 Clock Synchronization Sequence

In the case where TWI_THR or TWI_RHR is not written/read in time, TWI performs a clock synchronization.

Clock stretching information is given by the SCLWS (Clock Wait state) bit.

See Figure 25-28 on page 390 and Figure 25-29 on page 391.

25.9.4.4 General Call

In the case where a GENERAL CALL is performed, GACC (General Call ACCess) flag is set.

After GACC is set, it is up to the programmer to interpret the meaning of the GENERAL CALL and to decode the new address programming sequence.

See Figure 25-27 on page 389.

25.9.4.5 PDC

As it is impossible to know the exact number of data to receive/send, the use of PDC is NOT recommended in SLAVE mode.

25.9.5 Data Transfer

25.9.5.1 Read Operation

The read mode is defined as a data requirement from the master.

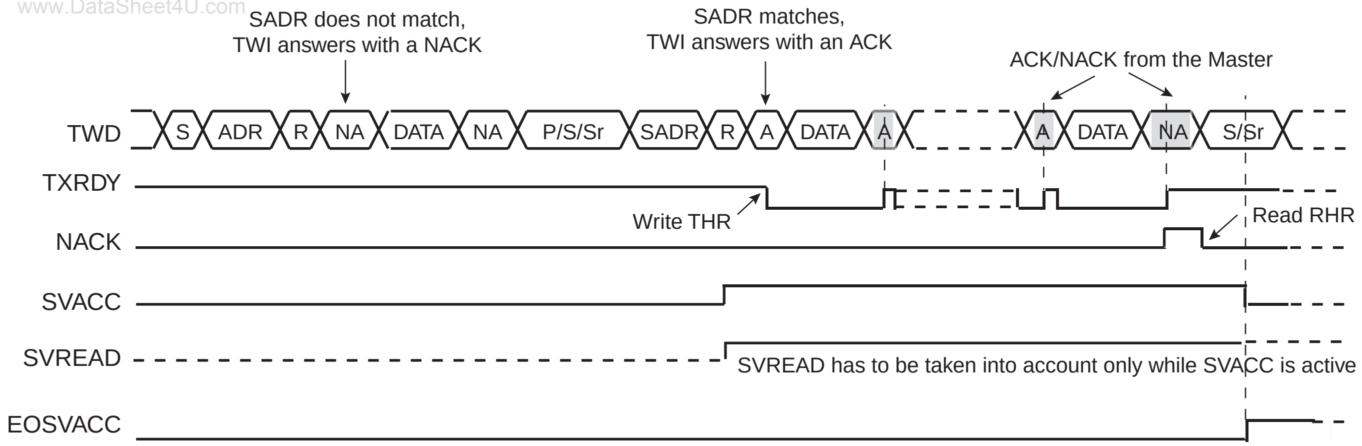
After a START or a REPEATED START condition is detected, the decoding of the address starts. If the slave address (SADR) is decoded, SVACC is set and SVREAD indicates the direction of the transfer.

Until a STOP or REPEATED START condition is detected, TWI continues sending data loaded in the TWI_THR register.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 25-25 on page 388 describes the write operation.

Figure 25-25. Read Access Ordered by a MASTER



Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
6. TXRDY is reset when data has been transmitted from TWI_THR to the shift register and set when this data has been acknowledged or non acknowledged.

25.9.5.2 Write Operation

The write mode is defined as a data transmission from the master.

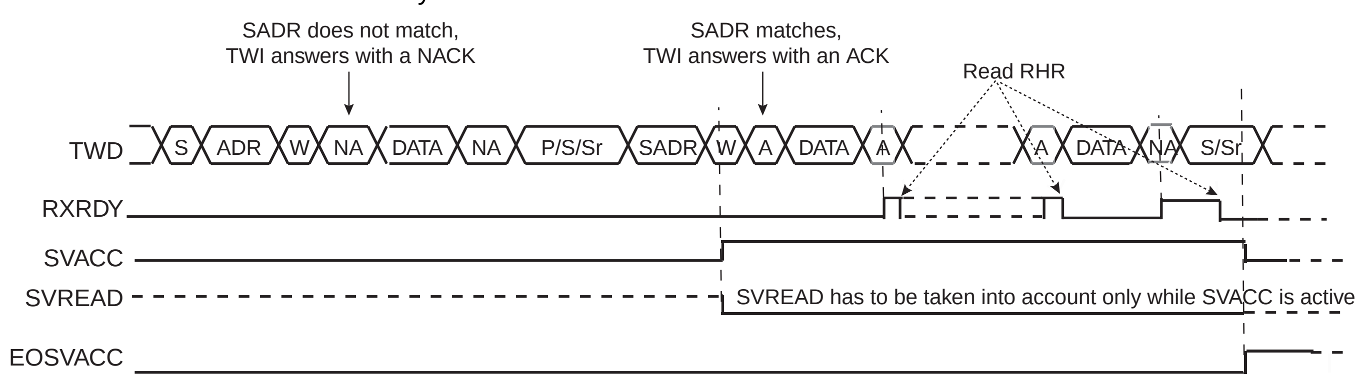
After a START or a REPEATED START, the decoding of the address starts. If the slave address is decoded, SVACC is set and SVREAD indicates the direction of the transfer (SVREAD is low in this case).

Until a STOP or REPEATED START condition is detected, TWI stores the received data in the TWI_RHR register.

If a STOP condition or a REPEATED START + an address different from SADR is detected, SVACC is reset.

Figure 25-26 on page 388 describes the Write operation.

Figure 25-26. Write Access Ordered by a Master



Notes:
1. When SVACC is low, the state of SVREAD becomes irrelevant.
2. RXRDY is set when data has been transmitted from the shift register to the TWI_RHR and reset when this data is read.

25.9.5.3 General Call

The general call is performed in order to change the address of the slave.

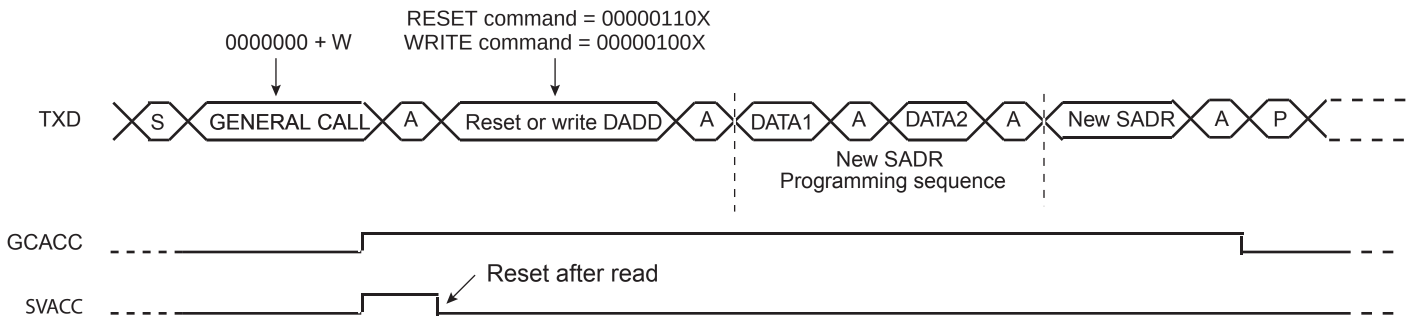
If a GENERAL CALL is detected, GACC is set.

After the detection of General Call, it is up to the programmer to decode the commands which come afterwards.

In case of a WRITE command, the programmer has to decode the programming sequence and program a new SADR if the programming sequence matches.

Figure 25-27 on page 389 describes the General Call access.

Figure 25-27. Master Performs a General Call



Note: This method allows the user to create an own programming sequence by choosing the programming bytes and the number of them. The programming sequence has to be provided to the master.

25.9.5.4 Clock Synchronization

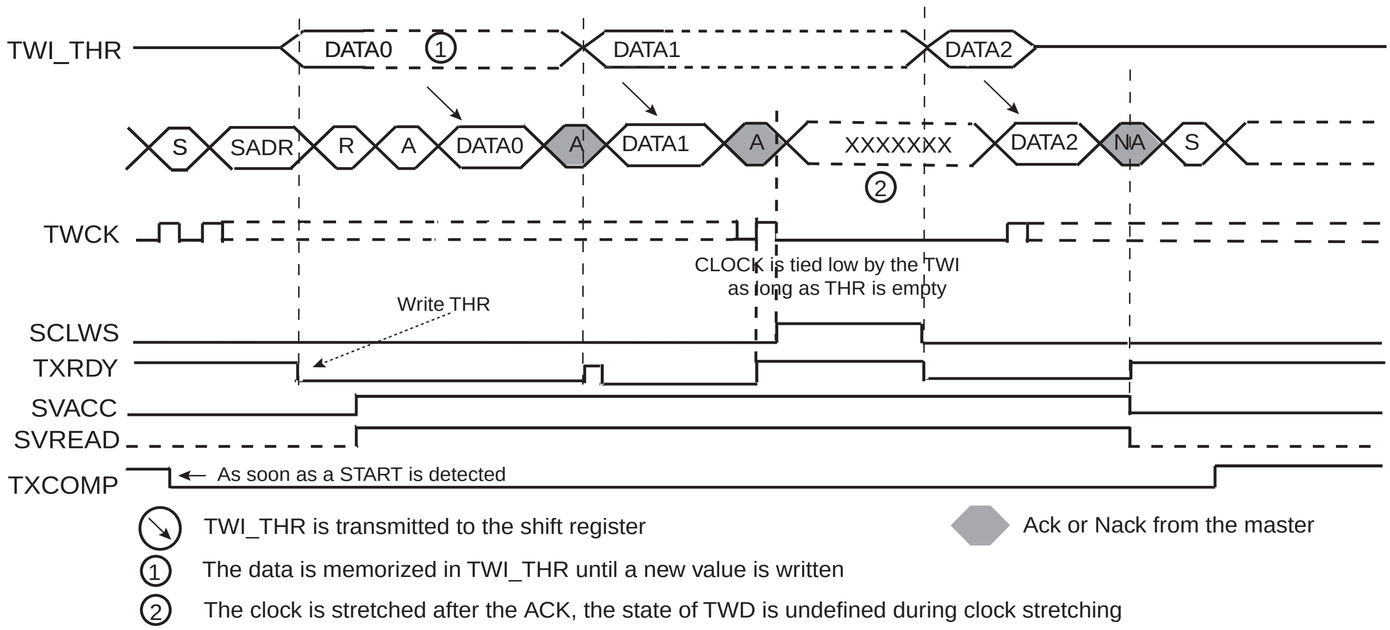
In both read and write modes, it may happen that TWI_THR/TWI_RHR buffer is not filled /emptied before the emission/reception of a new character. In this case, to avoid sending/receiving undesired data, a clock stretching mechanism is implemented.

25.9.5.4.1 Clock Synchronization in Read Mode

The clock is tied low if the shift register is empty and if a STOP or REPEATED START condition was not detected. It is tied low until the shift register is loaded.

Figure 25-28 on page 390 describes the clock synchronization in Read mode.

Figure 25-28. Clock Synchronization in Read Mode



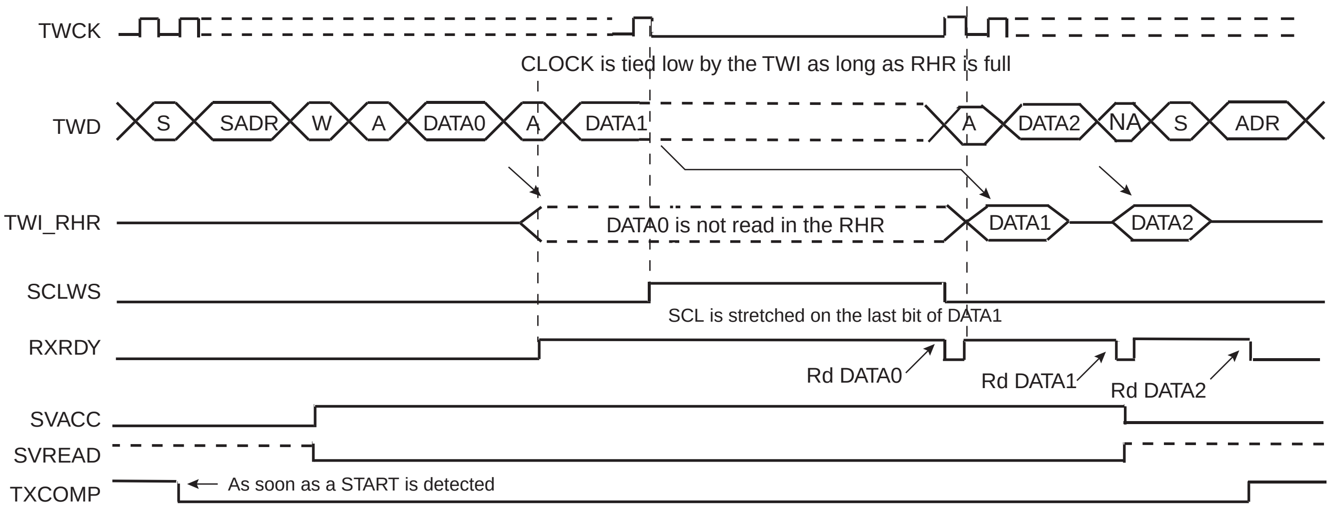
Notes:
1. TXRDY is reset when data has been written in the TWI_THR to the shift register and set when this data has been acknowledged or non acknowledged.
2. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.
3. SCLWS is automatically set when the clock synchronization mechanism is started.

25.9.5.4.1 Clock Synchronization in Write Mode

The clock is tied low if the shift register and the TWI_RHR is full. If a STOP or REPEATED_START condition was not detected, it is tied low until TWI_RHR is read.

Figure 25-29 on page 391 describes the clock synchronization in Read mode.

Figure 25-29. Clock Synchronization in Write Mode



Notes: 1. At the end of the read sequence, TXCOMP is set after a STOP or after a REPEATED_START + an address different from SADR.

2. SCLWS is automatically set when the clock synchronization mechanism is started and automatically reset when the mechanism is finished.

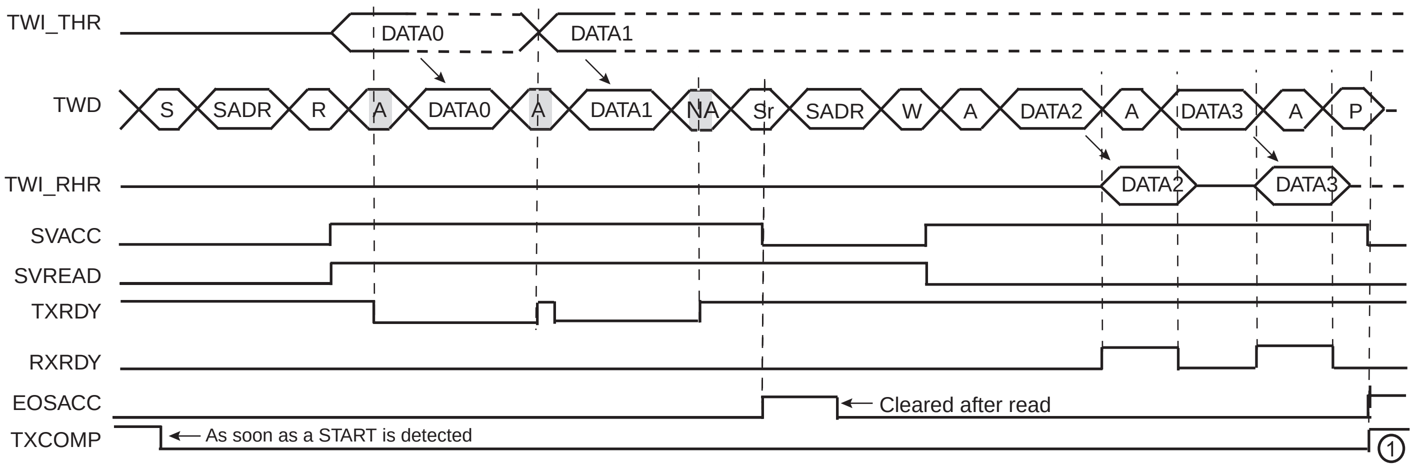
25.9.5.5 *Reversal after a Repeated Start*

25.9.5.5.1 *Reversal of Read to Write*

The master initiates the communication by a read command and finishes it by a write command.

Figure 25-30 on page 392 describes the repeated start + reversal from Read to Write mode.

Figure 25-30. Repeated Start + Reversal from Read to Write Mode

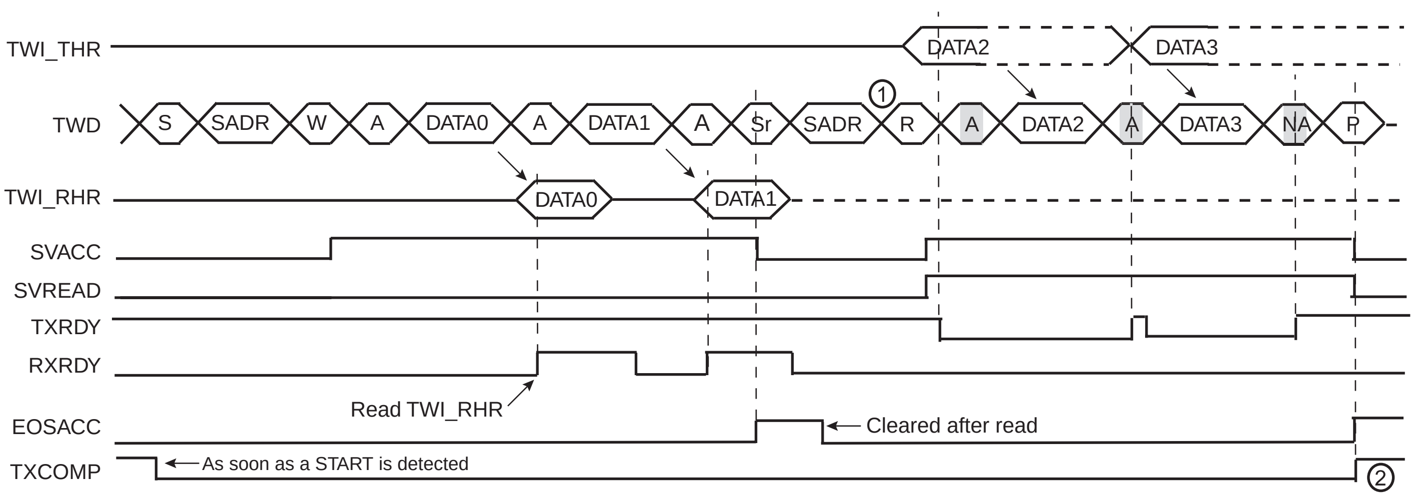


1. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

25.9.5.5.1 *Reversal of Write to Read*

The master initiates the communication by a write command and finishes it by a read command.Figure 25-31 on page 392 describes the repeated start + reversal from Write to Read mode.

Figure 25-31. Repeated Start + Reversal from Write to Read Mode



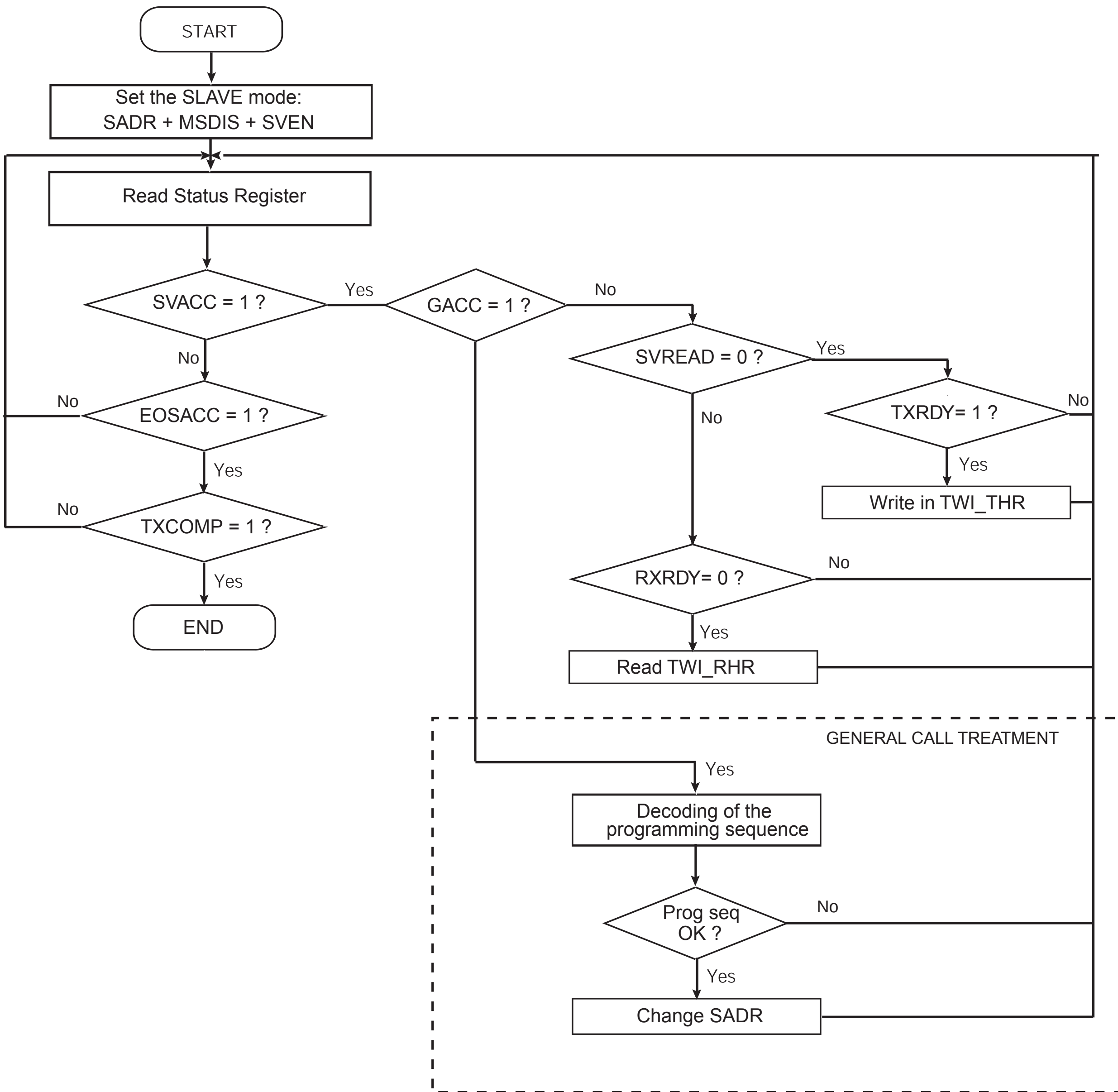
Notes: 1. In this case, if TWI_THR has not been written at the end of the read command, the clock is automatically stretched before the ACK.

7. TXCOMP is only set at the end of the transmission because after the repeated start, SADR is detected again.

25.9.6 Read Write Flowcharts

The flowchart shown in Figure 25-32 on page 393 gives an example of read and write operations in Slave mode. A polling or interrupt method can be used to check the status bits. The interrupt method requires that the interrupt enable register (TWI_IER) be configured first.

Figure 25-32. Read Write Flowchart in Slave Mode



25.10 Two-wire Interface (TWI) User Interface

Table 25-4. Register Mapping

Offset	Register	Name	Access	Reset
0x00	Control Register	TWI_CR	Write-only	N / A
0x04	Master Mode Register	TWI_MMR	Read-write	0x00000000
0x08	Slave Mode Register	TWI_SMR	Read-write	0x00000000
0x0C	Internal Address Register	TWI_IADR	Read-write	0x00000000
0x10	Clock Waveform Generator Register	TWI_CWGR	Read-write	0x00000000
0x20	Status Register	TWI_SR	Read-only	0x0000F009
0x24	Interrupt Enable Register	TWI_IER	Write-only	N / A
0x28	Interrupt Disable Register	TWI_IDR	Write-only	N / A
0x2C	Interrupt Mask Register	TWI_IMR	Read-only	0x00000000
0x30	Receive Holding Register	TWI_RHR	Read-only	0x00000000
0x34	Transmit Holding Register	TWI_THR	Write-only	0x00000000
0x38 - 0xFC	Reserved	–	–	–
0x100 - 0x124	Reserved for the PDC	–	–	–

25.10.1 TWI Control Register

Name: TWI_CR

Access: Write-only

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
SWRST	QUICK	SVDIS	SVEN	MSDIS	MSEN	STOP	START

- **START: Send a START Condition**

0 = No effect.

1 = A frame beginning with a START bit is transmitted according to the features defined in the mode register.

This action is necessary when the TWI peripheral wants to read data from a slave. When configured in Master Mode with a write operation, a frame is sent as soon as the user writes a character in the Transmit Holding Register (TWI_THR).

- **STOP: Send a STOP Condition**

0 = No effect.

1 = STOP Condition is sent just after completing the current byte transmission in master read mode.

  – In single data byte master read, the START and STOP must both be set.
  – In multiple data bytes master read, the STOP must be set after the last data received but one.
  – In master read mode, if a NACK bit is received, the STOP is automatically performed.
  – In master data write operation, a STOP condition will be sent after the transmission of the current data is finished.

- **MSEN: TWI Master Mode Enabled**

0 = No effect.

1 = If MSDIS = 0, the master mode is enabled.

Note: Switching from Slave to Master mode is only permitted when TXCOMP = 1.

- **MSDIS: TWI Master Mode Disabled**

0 = No effect.

1 = The master mode is disabled, all pending data is transmitted. The shifter and holding characters (if it contains data) are transmitted in case of write operation. In read operation, the character being transferred must be completely received before disabling.

- **SVEN: TWI Slave Mode Enabled**

0 = No effect.

1 = If SVDIS = 0, the slave mode is enabled.

Note: Switching from Master to Slave mode is only permitted when TXCOMP = 1.

- **SVDIS: TWI Slave Mode Disabled**

0 = No effect.

1 = The slave mode is disabled. The shifter and holding characters (if it contains data) are transmitted in case of read operation. In write operation, the character being transferred must be completely received before disabling.

- **QUICK: SMBUS Quick Command**

0 = No effect.

1 = If Master mode is enabled, a SMBUS Quick Command is sent.

- **SWRST: Software Reset**

0 = No effect.

1 = Equivalent to a system reset.

25.10.2 TWI Master Mode Register

Name: TWI_MMR

Access: Read-write

Reset Value: 0x00000000

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

23	22	21	20	19	18	17	16
–	DADR						

15	14	13	12	11	10	9	8
–	–	–	MREAD	–	–	IADRSZ	

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **IADRSZ: Internal Device Address Size**

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 to access slave devices in read or write mode. Those bits are only used in Master mode.

25.10.3 TWI Slave Mode Register

Name: TWI_SMR

Access: Read-write

Reset Value: 0x00000000

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

23	22	21	20	19	18	17	16
–	SADR						

15	14	13	12	11	10	9	8
–	–	–	–	–	–		

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **SADR: Slave Address**

The slave device address is used in Slave mode in order to be accessed by master devices in read or write mode.

SADR must be programmed before enabling the Slave mode or after a general call. Writes at other times have no effect.

25.10.4 TWI Internal Address Register

Name: TWI_IADR

Access: Read-write

Reset Value: 0x00000000

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

23	22	21	20	19	18	17	16
IADR							

15	14	13	12	11	10	9	8
IADR							

7	6	5	4	3	2	1	0
IADR							

- **IADR: Internal Address**

0, 1, 2 or 3 bytes depending on IADRSZ.

25.10.5 TWI Clock Waveform Generator Register

Name: TWI_CWGR

Access: Read-write

Reset Value: 0x00000000

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

23	22	21	20	19	18	17	16
					CKDIV		

15	14	13	12	11	10	9	8
CHDIV							

7	6	5	4	3	2	1	0
CLDIV							

TWI_CWGR is only used in Master mode.

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 4) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 4) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

25.10.6 TWI Status Register

Name: TWI_SR

Access: Read-only

Reset Value: 0x0000F009

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

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

15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCLWS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	SVREAD	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed (automatically set / reset)**

TXCOMP used in Master mode:

0 = During the length of the current frame.

1 = When both holding and shifter registers are empty and STOP condition has been sent.

TXCOMP behavior in Master mode can be seen in Figure 25-8 on page 373 and in Figure 25-10 on page 374.

TXCOMP used in Slave mode:

0 = As soon as a Start is detected.

1 = After a Stop or a Repeated Start + an address different from SADR is detected.

TXCOMP behavior in Slave mode can be seen in Figure 25-28 on page 390, Figure 25-29 on page 391, Figure 25-30 on page 392 and Figure 25-31 on page 392.

- **RXRDY: Receive Holding Register Ready (automatically set / reset)**

0 = No character has been received since the last TWI_RHR read operation.

1 = A byte has been received in the TWI_RHR since the last read.

RXRDY behavior in Master mode can be seen in Figure 25-10 on page 374.

RXRDY behavior in Slave mode can be seen in Figure 25-26 on page 388, Figure 25-29 on page 391, Figure 25-30 on page 392 and Figure 25-31 on page 392.

- **TXRDY: Transmit Holding Register Ready (automatically set / reset)**

TXRDY used in Master mode:

0 = The transmit holding register has not been transferred into shift register. Set to 0 when writing into TWI_THR register.

1 = As soon as a data byte is transferred from TWI_THR to internal shifter or if a NACK error is detected, TXRDY is set at the same time as TXCOMP and NACK. TXRDY is also set when MSEN is set (enable TWI).

TXRDY behavior in Master mode can be seen in Figure 25-8 on page 373.

TXRDY used in Slave mode:

0 = As soon as data is written in the TWI_THR, until this data has been transmitted and acknowledged (ACK or NACK).

1 = It indicates that the TWI_THR is empty and that data has been transmitted and acknowledged.

If TXRDY is high and if a NACK has been detected, the transmission will be stopped. Thus when TRDY = NACK = 1, the programmer must not fill TWI_THR to avoid losing it.

TXRDY behavior in Slave mode can be seen in Figure 25-25 on page 388, Figure 25-28 on page 390, Figure 25-30 on page 392 and Figure 25-31 on page 392.

- **SVREAD: Slave Read (automatically set / reset)**

This bit is only used in Slave mode. When SVACC is low (no Slave access has been detected) SVREAD is irrelevant.

0 = Indicates that a write access is performed by a Master.

1 = Indicates that a read access is performed by a Master.

SVREAD behavior can be seen in Figure 25-25 on page 388, Figure 25-26 on page 388, Figure 25-30 on page 392 and Figure 25-31 on page 392.

- **SVACC: Slave Access (automatically set / reset)**

This bit is only used in Slave mode.

0 = TWI is not addressed. SVACC is automatically cleared after a NACK or a STOP condition is detected.

1 = Indicates that the address decoding sequence has matched (A Master has sent SADR). SVACC remains high until a NACK or a STOP condition is detected.

SVACC behavior can be seen in Figure 25-25 on page 388, Figure 25-26 on page 388, Figure 25-30 on page 392 and Figure 25-31 on page 392.

- **GACC: General Call Access (clear on read)**

This bit is only used in Slave mode.

0 = No General Call has been detected.

1 = A General Call has been detected. After the detection of General Call, the programmer decoded the commands that follow and the programming sequence.

GACC behavior can be seen in Figure 25-27 on page 389.

- **OVRE: Overrun Error (clear on read)**

This bit is only used in Master mode.

0 = TWI_RHR has not been loaded while RXRDY was set

1 = TWI_RHR has been loaded while RXRDY was set. Reset by read in TWI_SR when TXCOMP is set.

- **NACK: Not Acknowledged (clear on read)**

NACK used in Master mode:

0 = Each data byte has been correctly received by the far-end side TWI slave component.

1 = A data byte has not been acknowledged by the slave component. Set at the same time as TXCOMP.

NACK used in Slave Read mode:

0 = Each data byte has been correctly received by the Master.

1 = In read mode, a data byte has not been acknowledged by the Master. When NACK is set the programmer must not fill TWI_THR even if TXRDY is set, because it means that the Master will stop the data transfer or re initiate it.

Note that in Slave Write mode all data are acknowledged by the TWI.

- **ARBLST: Arbitration Lost (clear on read)**

This bit is only used in Master mode.

0: Arbitration won.

1: Arbitration lost. Another master of the TWI bus has won the multi-master arbitration. TXCOMP is set at the same time.

- **SCLWS: Clock Wait State (automatically set / reset)**

This bit is only used in Slave mode.

0 = The clock is not stretched.

1 = The clock is stretched. TWI_THR / TWI_RHR buffer is not filled / emptied before the emission / reception of a new character.

SCLWS behavior can be seen in Figure 25-28 on page 390 and Figure 25-29 on page 391.

- **EOSACC: End Of Slave Access (clear on read)**

This bit is only used in Slave mode.

0 = A slave access is being performing.

1 = The Slave Access is finished. End Of Slave Access is automatically set as soon as SVACC is reset.

EOSACC behavior can be seen in Figure 25-30 on page 392 and Figure 25-31 on page 392

- **ENDRX: End of RX buffer**

This bit is only used in Master mode.

0 = The Receive Counter Register has not reached 0 since the last write in TWI_RCR or TWI_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in TWI_RCR or TWI_RNCR.

- **ENDTX: End of TX buffer**

This bit is only used in Master mode.

0 = The Transmit Counter Register has not reached 0 since the last write in TWI_TCR or TWI_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in TWI_TCR or TWI_TNCR.

- **RXBUFF: RX Buffer Full**

This bit is only used in Master mode.

0 = TWI_RCR or TWI_RNCR have a value other than 0.

1 = Both TWI_RCR and TWI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

This bit is only used in Master mode.

0 = TWI_TCR or TWI_TNCR have a value other than 0.

1 = Both TWI_TCR and TWI_TNCR have a value of 0.

25.10.7 TWI Interrupt Enable Register

Name: TWI_IER

Access: Write-only

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed Interrupt Enable**
- **RXRDY: Receive Holding Register Ready Interrupt Enable**
- **TXRDY: Transmit Holding Register Ready Interrupt Enable**
- **SVACC: Slave Access Interrupt Enable**
- **GACC: General Call Access Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **NACK: Not Acknowledge Interrupt Enable**
- **ARBLST: Arbitration Lost Interrupt Enable**
- **SCL_WS: Clock Wait State Interrupt Enable**
- **EOSACC: End Of Slave Access Interrupt Enable**
- **ENDRX: End of Receive Buffer Interrupt Enable**
- **ENDTX: End of Transmit Buffer Interrupt Enable**
- **RXBUFF: Receive Buffer Full Interrupt Enable**
- **TXBUFE: Transmit Buffer Empty Interrupt Enable**

0 = No effect.

1 = Enables the corresponding interrupt.

25.10.8 TWI Interrupt Disable Register

Name: TWI_IDR

Access: Write-only

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed Interrupt Disable**
- **RXRDY: Receive Holding Register Ready Interrupt Disable**
- **TXRDY: Transmit Holding Register Ready Interrupt Disable**
- **SVACC: Slave Access Interrupt Disable**
- **GACC: General Call Access Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **NACK: Not Acknowledge Interrupt Disable**
- **ARBLST: Arbitration Lost Interrupt Disable**
- **SCL_WS: Clock Wait State Interrupt Disable**
- **EOSACC: End Of Slave Access Interrupt Disable**
- **ENDRX: End of Receive Buffer Interrupt Disable**
- **ENDTX: End of Transmit Buffer Interrupt Disable**
- **RXBUFF: Receive Buffer Full Interrupt Disable**
- **TXBUFE: Transmit Buffer Empty Interrupt Disable**

0 = No effect.

1 = Disables the corresponding interrupt.

25.10.9 TWI Interrupt Mask Register

Name: TWI_IMR

Access: Read-only

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
TXBUFE	RXBUFF	ENDTX	ENDRX	EOSACC	SCL_WS	ARBLST	NACK

7	6	5	4	3	2	1	0
–	OVRE	GACC	SVACC	–	TXRDY	RXRDY	TXCOMP

- **TXCOMP: Transmission Completed Interrupt Mask**
- **RXRDY: Receive Holding Register Ready Interrupt Mask**
- **TXRDY: Transmit Holding Register Ready Interrupt Mask**
- **SVACC: Slave Access Interrupt Mask**
- **GACC: General Call Access Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **NACK: Not Acknowledge Interrupt Mask**
- **ARBLST: Arbitration Lost Interrupt Mask**
- **SCL_WS: Clock Wait State Interrupt Mask**
- **EOSACC: End Of Slave Access Interrupt Mask**
- **ENDRX: End of Receive Buffer Interrupt Mask**
- **ENDTX: End of Transmit Buffer Interrupt Mask**
- **RXBUFF: Receive Buffer Full Interrupt Mask**
- **TXBUFE: Transmit Buffer Empty Interrupt Mask**

0 = The corresponding interrupt is disabled.

1 = The corresponding interrupt is enabled.

25.10.10 TWI Receive Holding Register

Name: TWI_RHR

Access: Read-only

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
RXDATA							

- **RXDATA: Master or Slave Receive Holding Data**

25.10.11 TWI Transmit Holding Register

Name: TWI_THR

Access: Read-write

Reset Value: 0x00000000

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
TXDATA							

- **TXDATA: Master or Slave Transmit Holding Data**

26. Universal Synchronous/Asynchronous Receiver/Transceiver

26.1 Description

The Universal Synchronous Asynchronous Receiver Transceiver (USART) provides one full duplex universal synchronous asynchronous serial link. Data frame format is widely programmable (data length, parity, number of stop bits) to support a maximum of standards. The receiver implements parity error, framing error and overrun error detection. The receiver time-out enables handling variable-length frames and the transmitter timeguard facilitates communications with slow remote devices. Multidrop communications are also supported through address bit handling in reception and transmission.

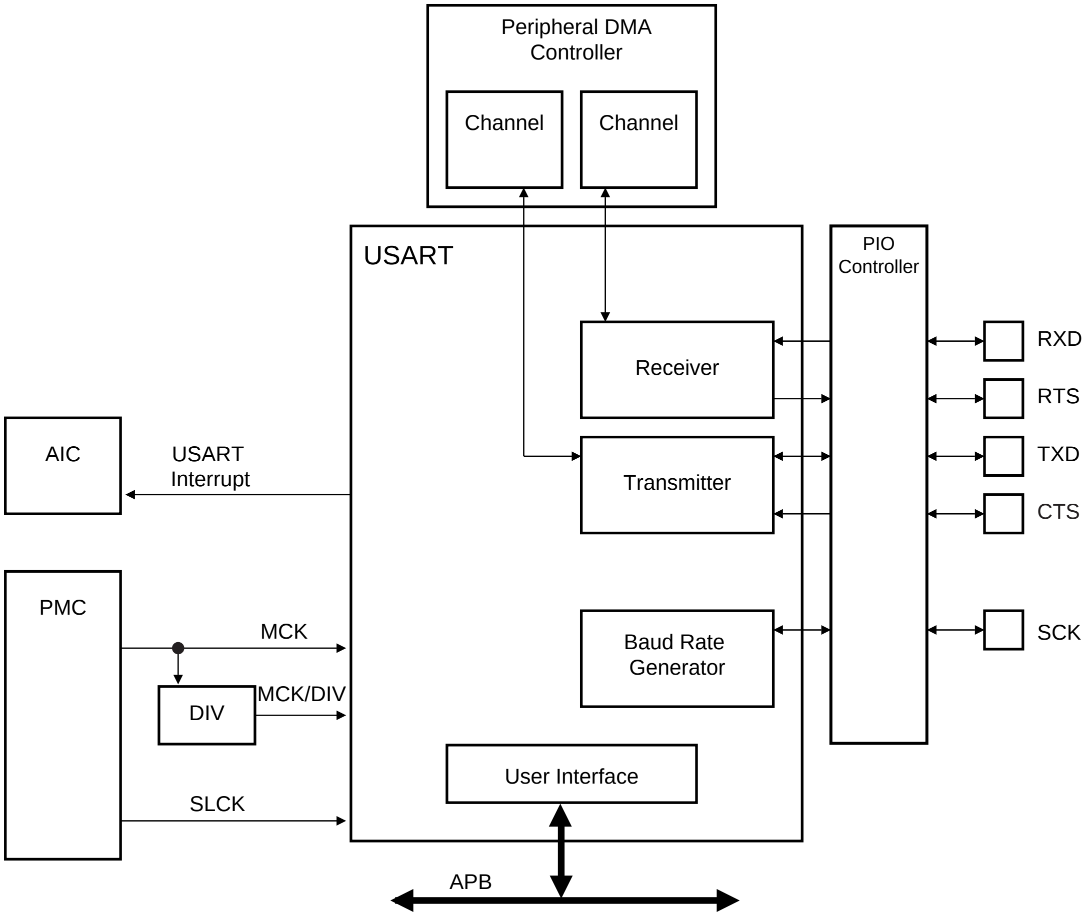
The USART features three test modes: remote loopback, local loopback and automatic echo.

The USART supports specific operating modes providing interfaces on RS485 buses, with ISO7816 T = 0 or T = 1 smart card slots and infrared transceivers. The hardware handshaking feature enables an out-of-band flow control by automatic management of the pins RTS and CTS.

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

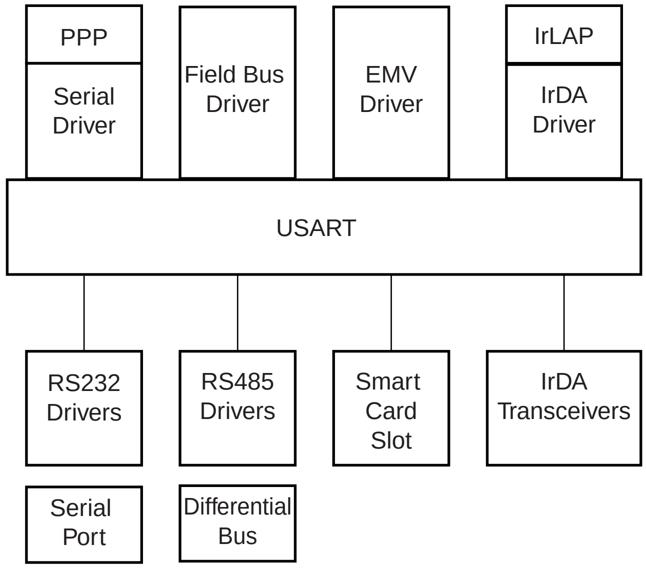
26.2 Block Diagram

Figure 26-1. USART Block Diagram



26.3 Application Block Diagram

Figure 26-2. Application Block Diagram



26.4 I/O Lines Description

Table 26-1. I/O Line Description

Name	Description	Type	Active Level
SCK	Serial Clock	I/O	
TXD	Transmit Serial Data	I/O	
RXD	Receive Serial Data	Input	
CTS	Clear to Send	Input	Low
RTS	Request to Send	Output	Low

26.5 Product Dependencies

26.5.1 I/O Lines

The pins used for interfacing the USART may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired USART pins to their peripheral function. If I/O lines of the USART are not used by the application, they can be used for other purposes by the PIO Controller.

To prevent the TXD line from falling when the USART is disabled, the use of an internal pull up is mandatory. If the hardware handshaking feature or Modem mode is used, the internal pull up on TXD must also be enabled.

26.5.2 Power Management

The USART is not continuously clocked. The programmer must first enable the USART Clock in the Power Management Controller (PMC) before using the USART. However, if the application does not require USART operations, the USART clock can be stopped when not needed and be restarted later. In this case, the USART will resume its operations where it left off.

Configuring the USART does not require the USART clock to be enabled.

26.5.3 Interrupt

The USART interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the USART interrupt requires the AIC to be programmed first. Note that it is not recommended to use the USART interrupt line in edge sensitive mode.

26.6 Functional Description

The USART is capable of managing several types of serial synchronous or asynchronous communications.

It supports the following communication modes:

- 5- to 9-bit full-duplex asynchronous serial communication
  - MSB- or LSB-first
  - 1, 1.5 or 2 stop bits
  - Parity even, odd, marked, space or none
  - By 8 or by 16 over-sampling receiver frequency
  - Optional hardware handshaking
  - Optional break management
  - Optional multidrop serial communication
- High-speed 5- to 9-bit full-duplex synchronous serial communication
  - MSB- or LSB-first
  - 1 or 2 stop bits
  - Parity even, odd, marked, space or none
  - By 8 or by 16 over-sampling frequency
  - Optional hardware handshaking
  - Optional break management
  - Optional multidrop serial communication
- RS485 with driver control signal
- ISO7816, T0 or T1 protocols for interfacing with smart cards
  - NACK handling, error counter with repetition and iteration limit
- InfraRed IrDA Modulation and Demodulation
- Test modes
  - Remote loopback, local loopback, automatic echo

26.6.1 Baud Rate Generator

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

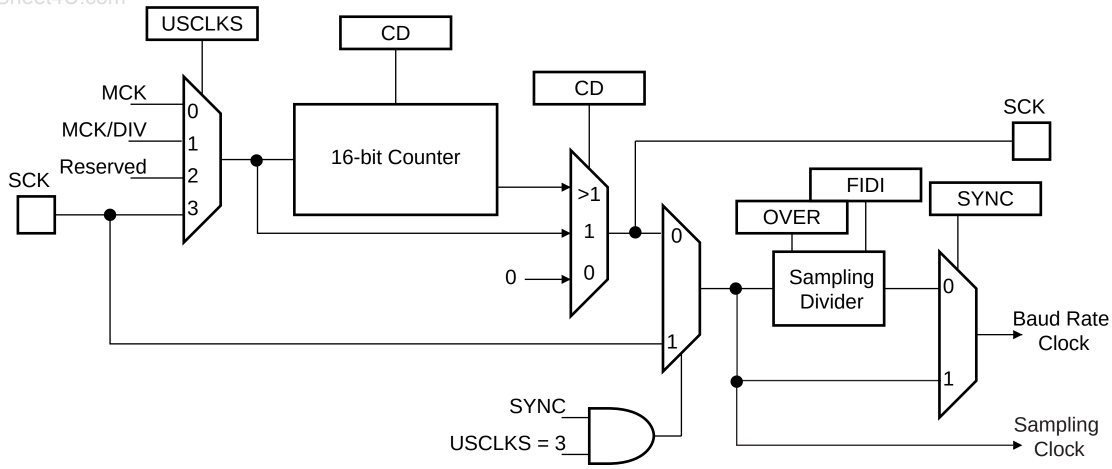
The Baud Rate Generator clock source can be selected by setting the USCLKS field in the Mode Register (US_MR) between:

- the Master Clock MCK
- a division of the Master Clock, the divider is 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 26-3. Baud Rate Generator



26.6.1.1 *Baud Rate in Asynchronous Mode*

If the USART is programmed to operate in asynchronous mode, the selected clock is first divided by CD, which is field programmed in the 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.

26.6.1.1.1 *Baud Rate Calculation Example*

Table 26-2 shows calculations of CD to obtain a baud rate at 38400 bauds for different source clock frequencies. This table also shows the actual resulting baud rate and the error.

Table 26-2. Baud Rate Example (OVER = 0)

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

Table 26-2. Baud Rate Example (OVER = 0) (Continued)

Source Clock	Expected Baud Rate	Calculation Result	CD	Actual Baud Rate	Error
14 318 180	38 400	23.30	23	38 908.10	1.31%
14 745 600	38 400	24.00	24	38 400.00	0.00%
18 432 000	38 400	30.00	30	38 400.00	0.00%
24 000 000	38 400	39.06	39	38 461.54	0.16%
24 576 000	38 400	40.00	40	38 400.00	0.00%
25 000 000	38 400	40.69	40	38 109.76	0.76%
32 000 000	38 400	52.08	52	38 461.54	0.16%
32 768 000	38 400	53.33	53	38 641.51	0.63%
33 000 000	38 400	53.71	54	38 194.44	0.54%
40 000 000	38 400	65.10	65	38 461.54	0.16%
50 000 000	38 400	81.38	81	38 580.25	0.47%

The baud rate is calculated with the following formula:

$$BaudRate = MCK / CD \times 16$$

The baud rate error is calculated with the following formula. It is not recommended to work with an error higher than 5%.

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

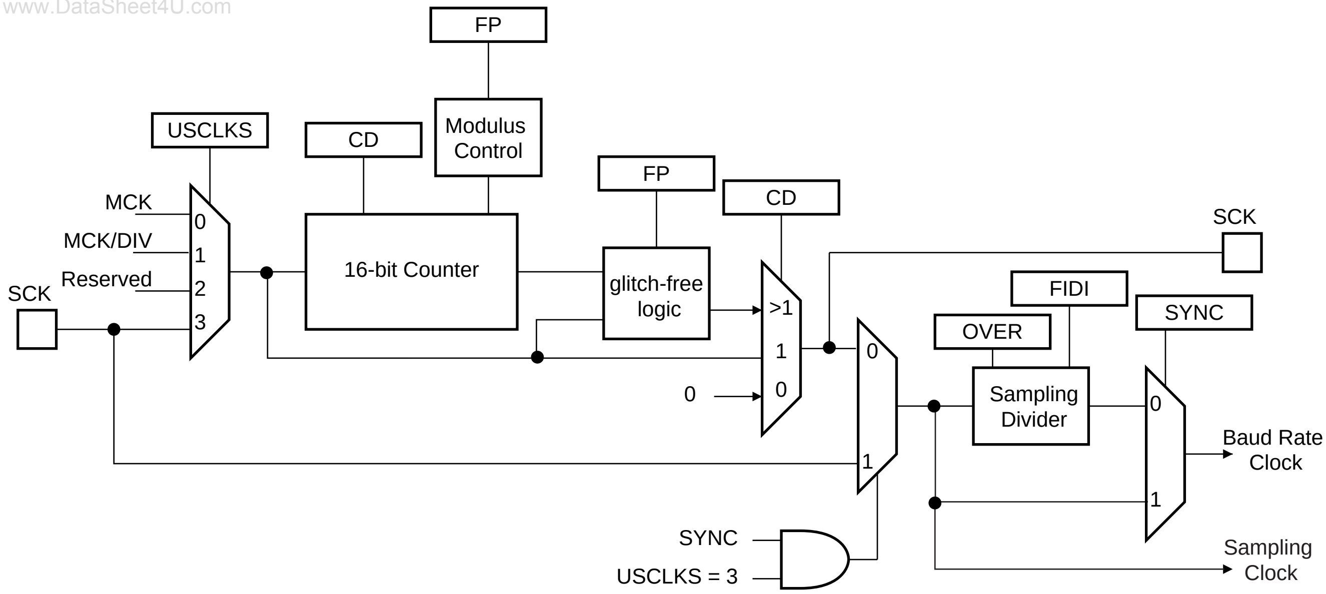
26.6.1.2 *Fractional Baud Rate in Asynchronous Mode*

The Baud Rate generator previously defined is subject to the following limitation: the output frequency changes by only integer multiples of the reference frequency. An approach to this problem is to integrate a fractional N clock generator that has a high resolution. The generator architecture is modified to obtain Baud Rate changes by a fraction of the reference source clock. This fractional part is programmed with the FP field in the Baud Rate Generator Register (US_BRGR). If FP is not 0, the fractional part is activated. The resolution is one eighth of the clock divider. This feature is only available when using USART normal mode. The fractional Baud Rate is calculated using the following formula:

$$Baudrate = \frac{SelectedClock}{\left(8(2 - Over)\left(CD + \frac{FP}{8}\right)\right)}$$

The modified architecture is presented below:

Figure 26-4. Fractional Baud Rate Generator



26.6.1.3 Baud Rate in Synchronous Mode

If the USART is programmed to operate in synchronous mode, the selected clock is simply divided by the field CD in US_BRGR.

$$BaudRate = \frac{SelectedClock}{CD}$$

In synchronous mode, if the external clock is selected (USCLKS = 3), the clock is provided directly by the signal on the USART SCK pin. No division is active. The value written in US_BRGR has no effect. The external clock frequency must be at least 4.5 times lower than the system clock.

When either the external clock SCK or the internal clock divided (MCK/DIV) is selected, the value programmed in CD must be even if the user has to ensure a 50:50 mark/space ratio on the SCK pin. If the internal clock MCK is selected, the Baud Rate Generator ensures a 50:50 duty cycle on the SCK pin, even if the value programmed in CD is odd.

26.6.1.4 Baud Rate in ISO 7816 Mode

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

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

where:

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

Di is a binary value encoded on a 4-bit field, named DI, as represented in Table 26-3.

Table 26-3. Binary and Decimal Values for Di

DI field	0001	0010	0011	0100	0101	0110	1000	1001
Di (decimal)	1	2	4	8	16	32	12	20

Fi is a binary value encoded on a 4-bit field, named FI, as represented in Table 26-4.

Table 26-4. Binary and Decimal Values for Fi

FI field	0000	0001	0010	0011	0100	0101	0110	1001	1010	1011	1100	1101
Fi (decimal	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048

Table 26-5 shows the resulting Fi/Di Ratio, which is the ratio between the ISO7816 clock and the baud rate clock.

Table 26-5. Possible Values for the Fi/Di Ratio

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

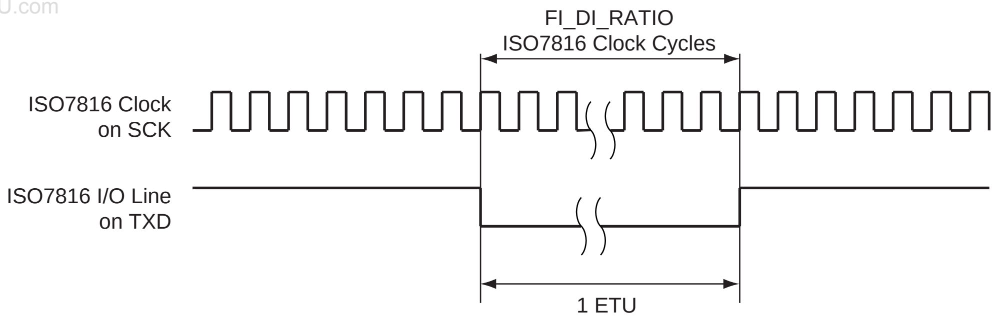
If the USART is configured in ISO7816 Mode, the clock selected by the USCLKS field in the Mode Register (US_MR) is first divided by the value programmed in the field CD in the Baud Rate Generator Register (US_BRGR). The resulting clock can be provided to the SCK pin to feed the smart card clock inputs.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

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

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

Figure 26-5 shows the relation between the Elementary Time Unit, corresponding to a bit time, and the ISO 7816 clock.

Figure 26-5. Elementary Time Unit (ETU)



26.6.2 Receiver and Transmitter Control

After reset, the receiver is disabled. The user must enable the receiver by setting the RXEN bit in the Control Register (US_CR). However, the receiver registers can be programmed before the receiver clock is enabled.

After reset, the transmitter is disabled. The user must enable it by setting the TXEN bit in the Control Register (US_CR). However, the transmitter registers can be programmed before being enabled.

The Receiver and the Transmitter can be enabled together or independently.

At any time, the software can perform a reset on the receiver or the transmitter of the USART by setting the corresponding bit, RSTRX and RSTTX respectively, in the Control Register (US_CR). The software resets clear the status flag and reset internal state machines but the user interface configuration registers hold the value configured prior to software reset. Regardless of what the receiver or the transmitter is performing, the communication is immediately stopped.

The user can also independently disable the receiver or the transmitter by setting RXDIS and TXDIS respectively in US_CR. If the receiver is disabled during a character reception, the USART waits until the end of reception of the current character, then the reception is stopped. If the transmitter is disabled while it is operating, the USART waits the end of transmission of both the current character and character being stored in the Transmit Holding Register (US_THR). If a timeguard is programmed, it is handled normally.

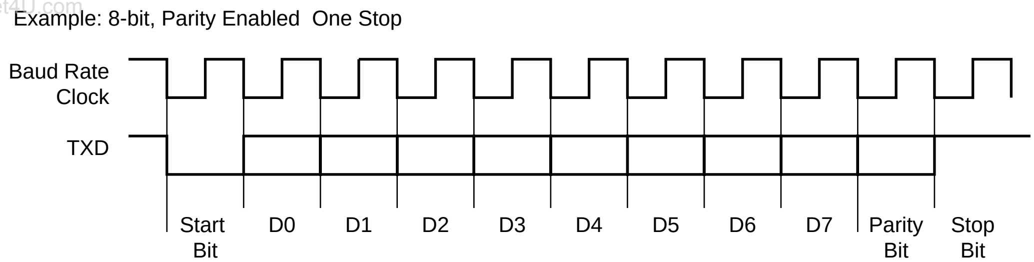
26.6.3 Synchronous and Asynchronous Modes

26.6.3.1 Transmitter Operations

The transmitter performs the same in both synchronous and asynchronous operating modes (SYNC = 0 or SYNC = 1). One start bit, up to 9 data bits, one optional parity bit and up to two stop bits are successively shifted out on the TXD pin at each falling edge of the programmed serial clock.

The number of data bits is selected by the CHRL field and the MODE 9 bit in 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.

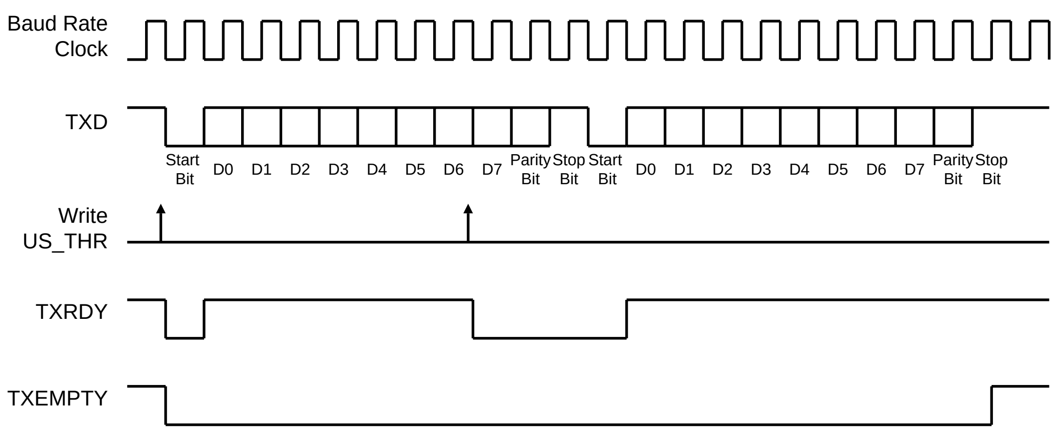
Figure 26-6. Character Transmit



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

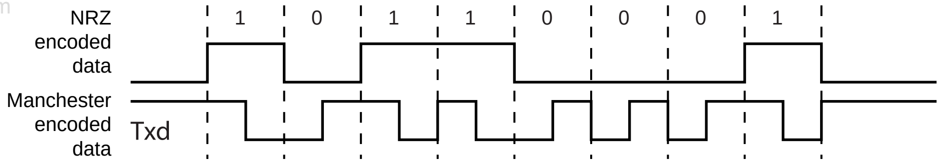
Figure 26-7. Transmitter Status



26.6.3.2 Manchester Encoder

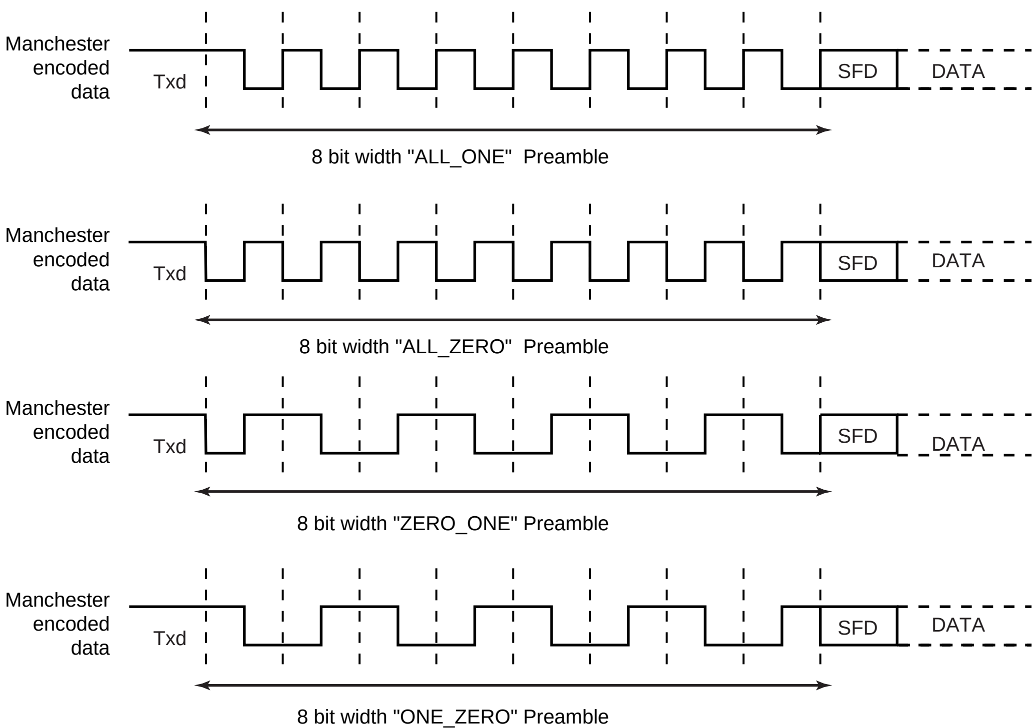
When the Manchester encoder is in use, characters transmitted through the USART are encoded based on biphase Manchester II format. To enable this mode, set the MAN field in the US_MR register to 1. Depending on polarity configuration, a logic level (zero or one), is transmitted as a coded signal one-to-zero or zero-to-one. Thus, a transition always occurs at the midpoint of each bit time. It consumes more bandwidth than the original NRZ signal (2x) but the receiver has more error control since the expected input must show a change at the center of a bit cell. An example of Manchester encoded sequence is: the byte 0xB1 or 10110001 encodes to 10 01 10 10 01 01 01 10, assuming the default polarity of the encoder. Figure 26-8 illustrates this coding scheme.

Figure 26-8. NRZ to Manchester Encoding



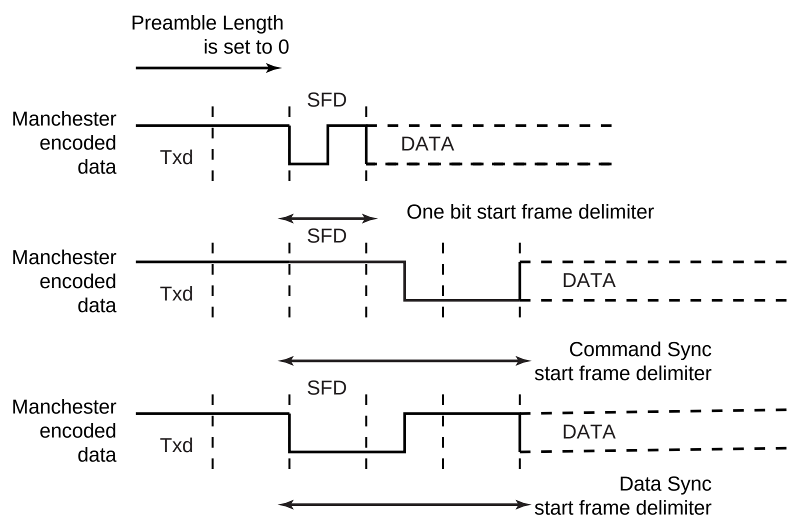
The Manchester encoded character can also be encapsulated by adding both a configurable preamble and a start frame delimiter pattern. Depending on the configuration, the preamble is a training sequence, composed of a pre-defined pattern with a programmable length from 1 to 15 bit times. If the preamble length is set to 0, the preamble waveform is not generated prior to any character. The preamble pattern is chosen among the following sequences: ALL_ONE, ALL_ZERO, ONE_ZERO or ZERO_ONE, writing the field TX_PP in the US_MAN register, the field TX_PL is used to configure the preamble length. Figure 26-9 illustrates and defines the valid patterns. To improve flexibility, the encoding scheme can be configured using the TX_MPOL field in the US_MAN register. If the TX_MPOL field is set to zero (default), a logic zero is encoded with a zero-to-one transition and a logic one is encoded with a one-to-zero transition. If the TX_MPOL field is set to one, a logic one is encoded with a one-to-zero transition and a logic zero is encoded with a zero-to-one transition.

Figure 26-9. Preamble Patterns, Default Polarity Assumed



A start frame delimiter is to be configured using the ONEBIT field in the US_MR register. It consists of a user-defined pattern that indicates the beginning of a valid data. Figure 26-10 illustrates these patterns. If the start frame delimiter, also known as start bit, is one bit, (ONEBIT at 1), a logic zero is Manchester encoded and indicates that a new character is being sent serially on the line. If the start frame delimiter is a synchronization pattern also referred to as sync (ONEBIT at 0), a sequence of 3 bit times is sent serially on the line to indicate the start of a new character. The sync waveform is in itself an invalid Manchester waveform as the transition occurs at the middle of the second bit time. Two distinct sync patterns are used: the command sync and the data sync. The command sync has a logic one level for one and a half bit times, then a transition to logic zero for the second one and a half bit times. If the MODSYNC field in the US_MR register is set to 1, the next character is a command. If it is set to 0, the next character is a data. When direct memory access is used, the MODSYNC field can be immediately updated with a modified character located in memory. To enable this mode, VAR_SYNC field in US_MR register must be set to 1. In this case, the MODSYNC field in US_MR is bypassed and the sync configuration is held in the TXSYNH in the US_THR register. The USART character format is modified and includes sync information.

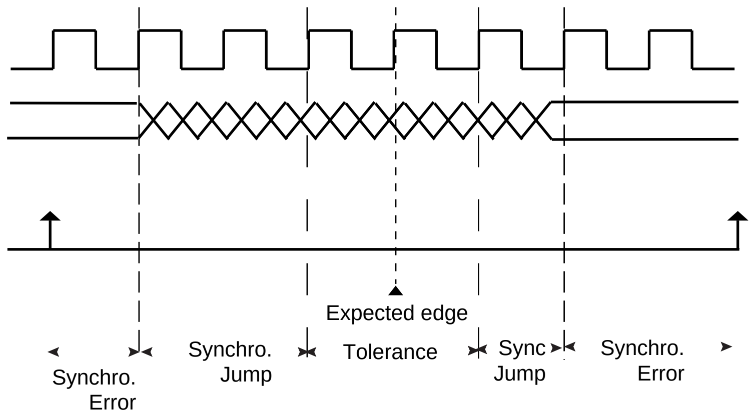
Figure 26-10. Start Frame Delimiter



26.6.3.2.1 Drift Compensation

Drift compensation is available only in 16X oversampling mode. An hardware recovery system allows a larger clock drift. To enable the hardware system, the bit in the USART_MAN register must be set. If the RXD edge is one 16X clock cycle from the expected edge, this is considered as normal jitter and no corrective actions is taken. If the RXD event is between 4 and 2 clock cycles before the expected edge, then the current period is shortened by one clock cycle. If the RXD event is between 2 and 3 clock cycles after the expected edge, then the current period is lengthened by one clock cycle. These intervals are considered to be drift and so corrective actions are automatically taken.

Figure 26-11. Bit Resynchronization



26.6.3.3 *Asynchronous Receiver*

If the USART is programmed in asynchronous operating mode (SYNC = 0), the receiver oversamples the RXD input line. The oversampling is either 16 or 8 times the Baud Rate clock, depending on the OVER bit in the Mode Register (US_MR).

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

If the oversampling is 16, (OVER at 0), a start is detected at the eighth sample at 0. Then, data bits, parity bit and stop bit are sampled on each 16 sampling clock cycle. If the oversampling is 8 (OVER at 1), a start bit is detected at the fourth sample at 0. Then, data bits, parity bit and stop bit are sampled on each 8 sampling clock cycle.

The number of data bits, first bit sent and parity mode are selected by the same fields and bits as the transmitter, i.e. respectively CHRL, MODE9, MSBF and PAR. For the synchronization mechanism **only**, the number of stop bits has no effect on the receiver as it considers only one stop bit, regardless of the field NBSTOP, so that resynchronization between the receiver and the transmitter can occur. Moreover, as soon as the stop bit is sampled, the receiver starts looking for a new start bit so that resynchronization can also be accomplished when the transmitter is operating with one stop bit.

Figure 26-12 and Figure 26-13 illustrate start detection and character reception when USART operates in asynchronous mode.

Figure 26-12. Asynchronous Start Detection

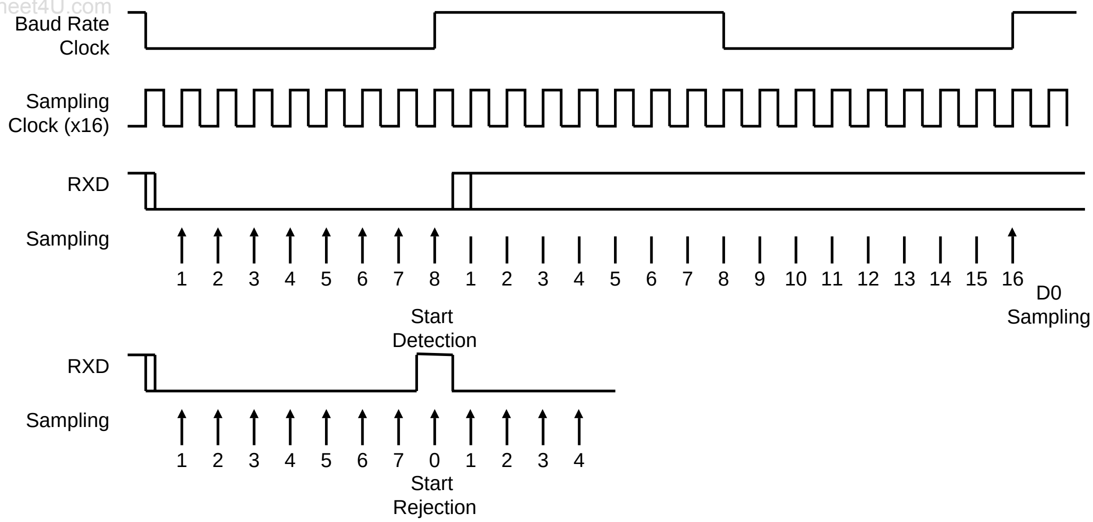
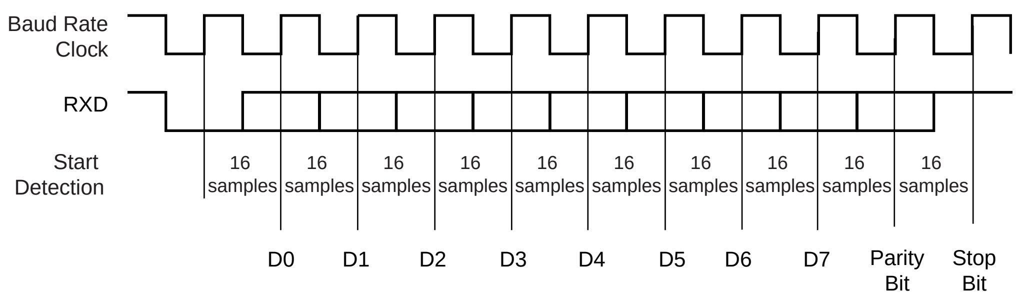


Figure 26-13. Asynchronous Character Reception



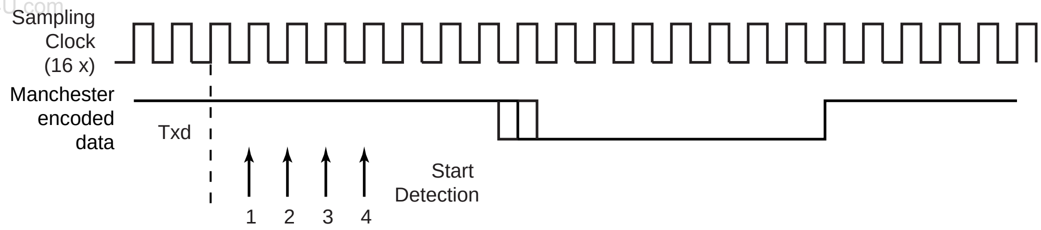
26.6.3.4 *Manchester Decoder*

When the MAN field in US_MR register is set to 1, the Manchester decoder is enabled. The decoder performs both preamble and start frame delimiter detection. One input line is dedicated to Manchester encoded input data.

An optional preamble sequence can be defined, its length is user-defined and totally independent of the emitter side. Use RX_PL in US_MAN register to configure the length of the preamble sequence. If the length is set to 0, no preamble is detected and the function is disabled. In addition, the polarity of the input stream is programmable with RX_MPOL field in US_MAN register. Depending on the desired application the preamble pattern matching is to be defined via the RX_PP field in US_MAN. See Figure 26-9 for available preamble patterns.

Unlike preamble, the start frame delimiter is shared between Manchester Encoder and Decoder. So, if ONEBIT field is set to 1, only a zero encoded Manchester can be detected as a valid start frame delimiter. If ONEBIT is set to 0, only a sync pattern is detected as a valid start frame delimiter. Decoder operates by detecting transition on incoming stream. If RXD is sampled during one quarter of a bit time at zero, a start bit is detected. See Figure 26-14. The sample pulse rejection mechanism applies.

Figure 26-14. Asynchronous Start Bit Detection



The receiver is activated and starts Preamble and Frame Delimiter detection, sampling the data at one quarter and then three quarters. If a valid preamble pattern or start frame delimiter is detected, the receiver continues decoding with the same synchronization. If the stream does not match a valid pattern or a valid start frame delimiter, the receiver re-synchronizes on the next valid edge.The minimum time threshold to estimate the bit value is three quarters of a bit time.

If a valid preamble (if used) followed with a valid start frame delimiter is detected, the incoming stream is decoded into NRZ data and passed to USART for processing. Figure 26-15 illustrates Manchester pattern mismatch. When incoming data stream is passed to the USART, the receiver is also able to detect Manchester code violation. A code violation is a lack of transition in the middle of a bit cell. In this case, MANE flag in US_CSR register is raised. It is cleared by writing the Control Register (US_CR) with the RSTSTA bit at 1. See Figure 26-16 for an example of Manchester error detection during data phase.

Figure 26-15. Preamble Pattern Mismatch

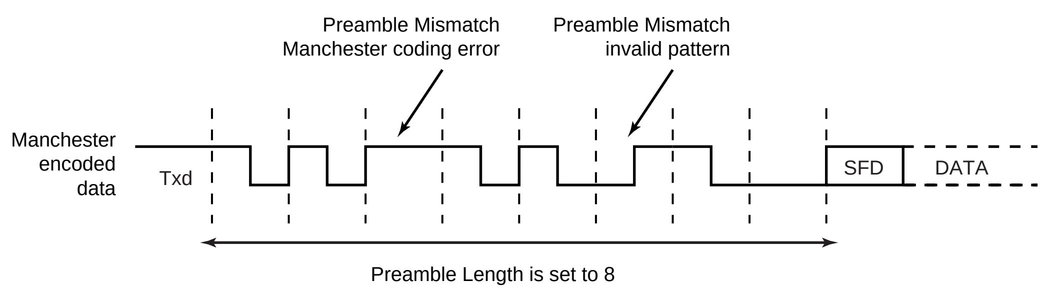
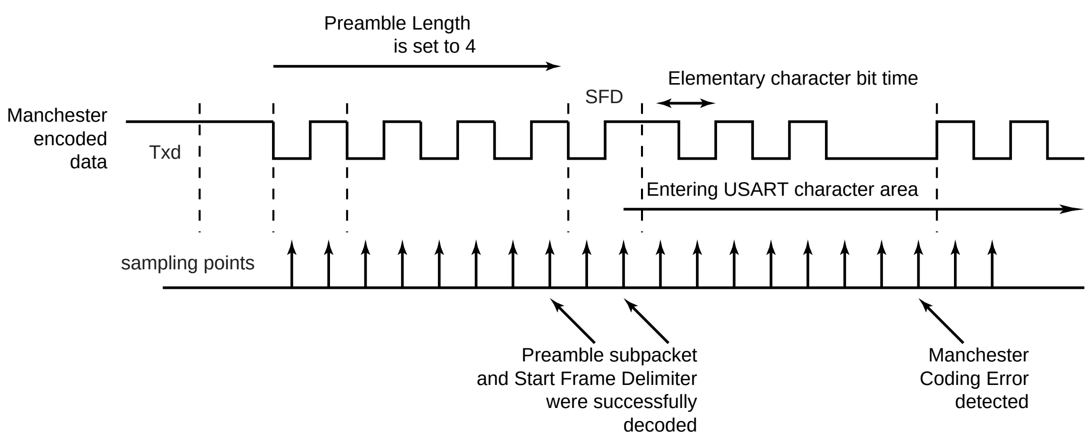


Figure 26-16. Manchester Error Flag



When the start frame delimiter is a sync pattern (ONEBIT field at 0), both command and data delimiter are supported. If a valid sync is detected, the received character is written as RXCHR field in the US_RHR register and the RXSYNH is updated. RXCHR is set to 1 when the received character is a command, and it is set to 0 if the received character is a data. This mechanism alleviates and simplifies the direct memory access as the character contains its own sync field in the same register.

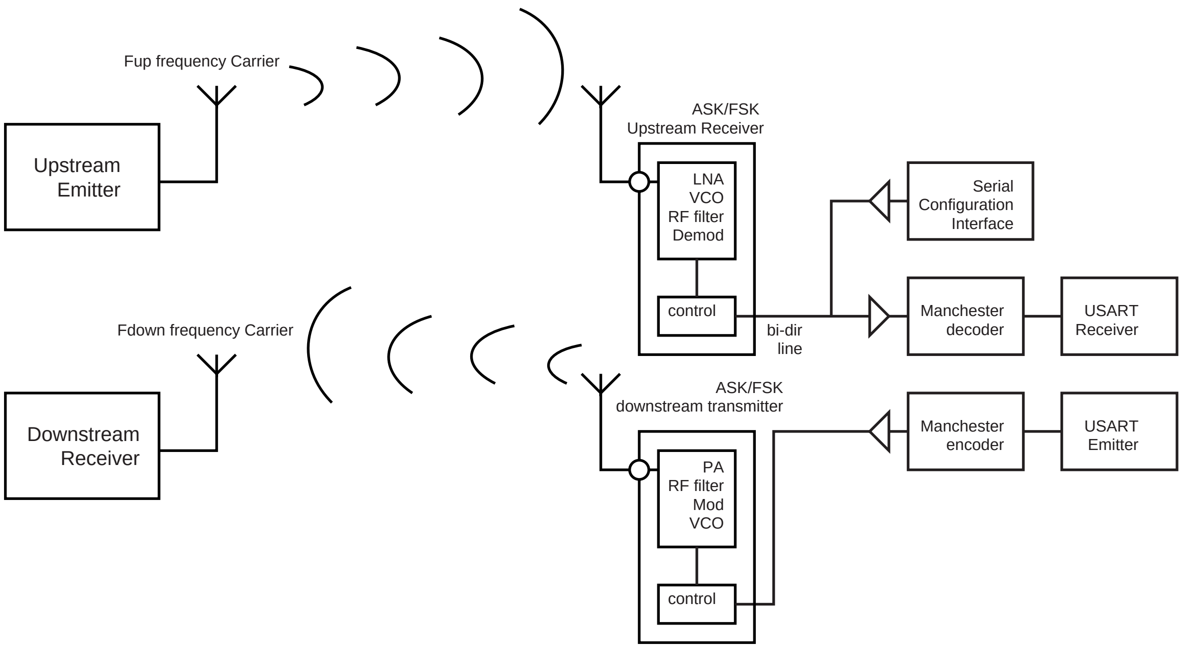
As the decoder is setup to be used in unipolar mode, the first bit of the frame has to be a zero-to-one transition.

26.6.3.5 *Radio Interface: Manchester Encoded USART Application*

This section describes low data rate RF transmission systems and their integration with a Manchester encoded USART. These systems are based on transmitter and receiver ICs that support ASK and FSK modulation schemes.

The goal is to perform full duplex radio transmission of characters using two different frequency carriers. See the configuration in Figure 26-17.

Figure 26-17. Manchester Encoded Characters RF Transmission



The USART module is configured as a Manchester encoder/decoder. Looking at the downstream communication channel, Manchester encoded characters are serially sent to the RF emitter. This may also include a user defined preamble and a start frame delimiter. Mostly, preamble is used in the RF receiver to distinguish between a valid data from a transmitter and signals due to noise. The Manchester stream is then modulated. See Figure 26-18 for an example of ASK modulation scheme. When a logic one is sent to the ASK modulator, the power amplifier, referred to as PA, is enabled and transmits an RF signal at downstream frequency. When a logic zero is transmitted, the RF signal is turned off. If the FSK modulator is activated, two different frequencies are used to transmit data. When a logic 1 is sent, the modulator outputs an RF signal at frequency F0 and switches to F1 if the data sent is a 0. See Figure 26-19.

From the receiver side, another carrier frequency is used. The RF receiver performs a bit check operation examining demodulated data stream. If a valid pattern is detected, the receiver switches to receiving mode. The demodulated stream is sent to the Manchester decoder. Because of bit checking inside RF IC, the data transferred to the microcontroller is reduced by a user-defined number of bits. The Manchester preamble length is to be defined in accordance with the RF IC configuration.

Figure 26-18. ASK Modulator Output

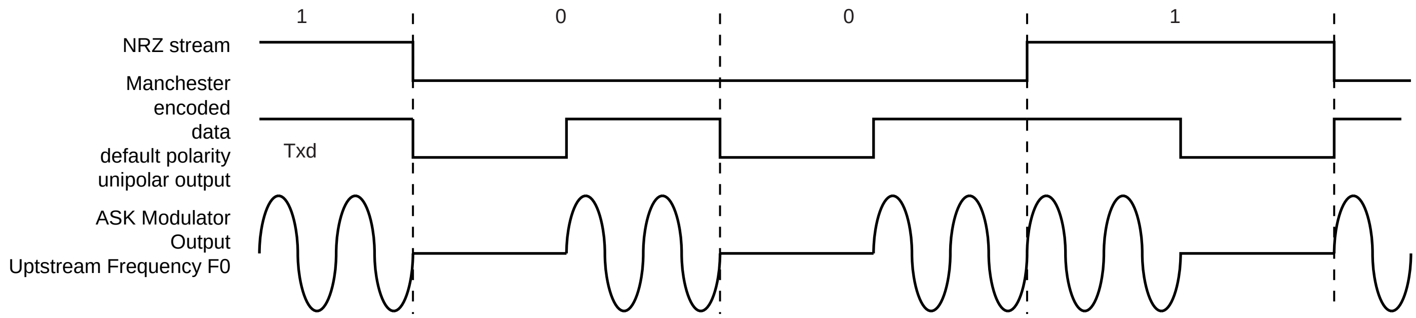
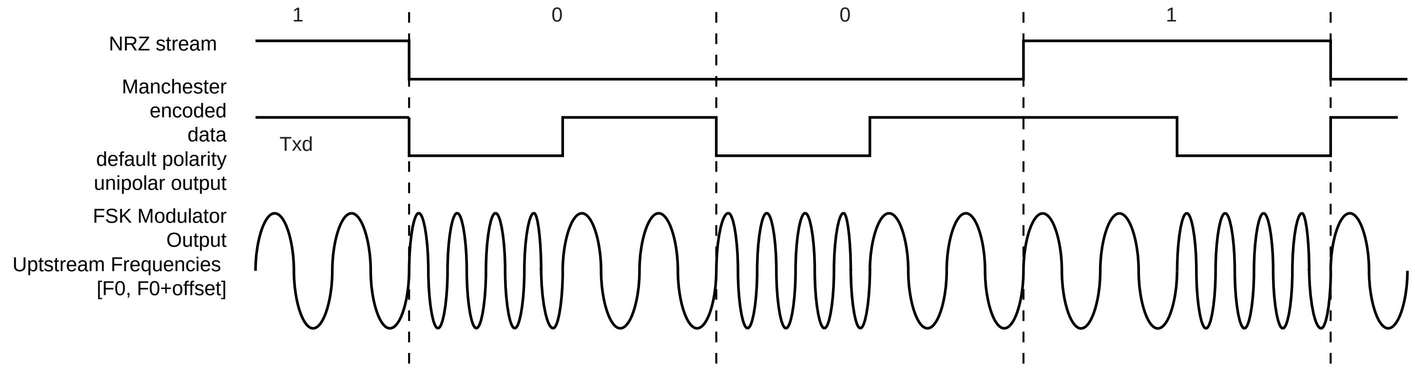


Figure 26-19. FSK Modulator Output



26.6.3.6 Synchronous Receiver

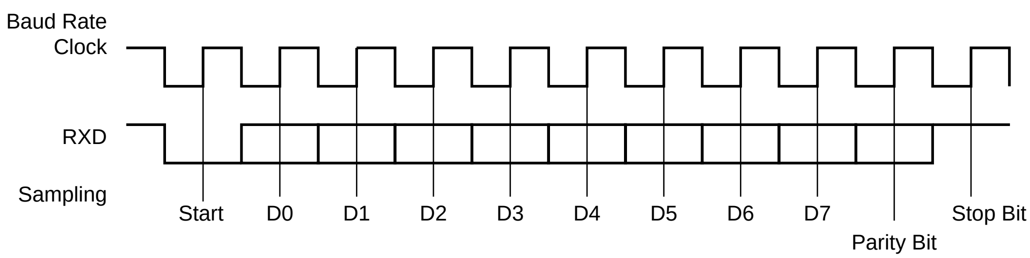
In synchronous mode (SYNC = 1), the receiver samples the RXD signal on each rising edge of the Baud Rate Clock. If a low level is detected, it is considered as a start. All data bits, the parity bit and the stop bits are sampled and the receiver waits for the next start bit. Synchronous mode operations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

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

Figure 26-20 illustrates a character reception in synchronous mode.

Figure 26-20. Synchronous Mode Character Reception

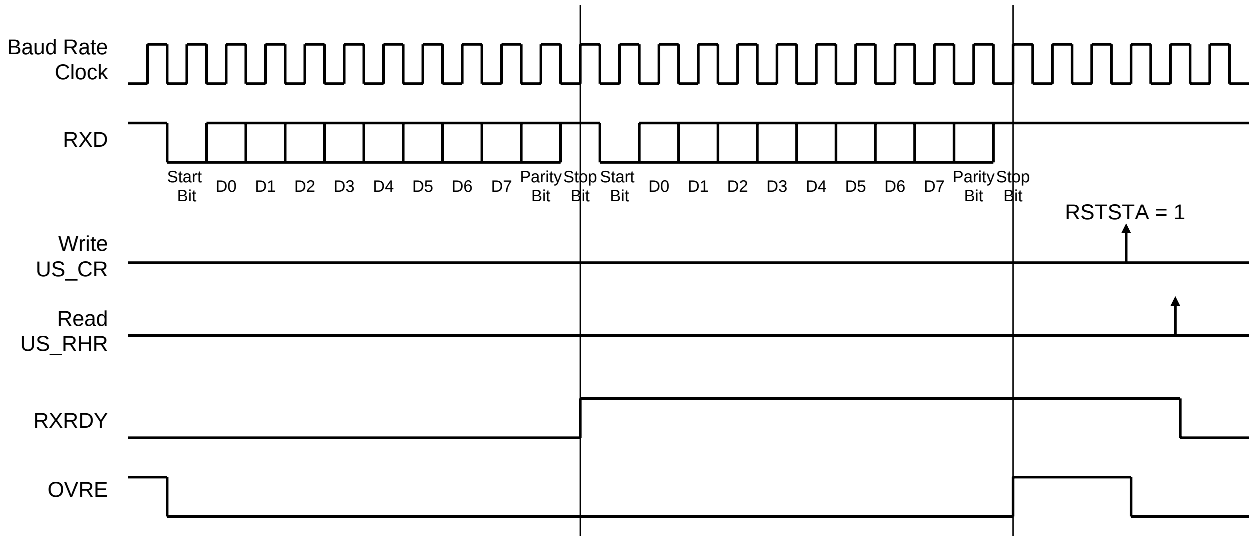
Example: 8-bit, Parity Enabled 1 Stop



26.6.3.7 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 26-21. Receiver Status



26.6.3.8 Parity

The USART supports five parity modes selected by programming the PAR field in the Mode Register (US_MR). The PAR field also enables the Multidrop mode, see "Multidrop Mode" on page 429. Even and odd parity bit generation and error detection are supported.

If even parity is selected, the parity generator of the transmitter drives the parity bit at 0 if a number of 1s in the character data bit is even, and at 1 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If odd parity is selected, the parity generator of the transmitter drives the parity bit at 1 if a number of 1s in the character data bit is even, and at 0 if the number of 1s is odd. Accordingly, the receiver parity checker counts the number of received 1s and reports a parity error if the sampled parity bit does not correspond. If the mark parity is used, the parity generator of the transmitter drives the parity bit at 1 for all characters. The receiver parity checker reports an error if the parity bit is sampled at 0. If the space parity is used, the parity generator of the transmitter drives the parity bit at 0 for all characters. The receiver parity checker reports an error if the parity bit is sampled at 1. If parity is disabled, the transmitter does not generate any parity bit and the receiver does not report any parity error.

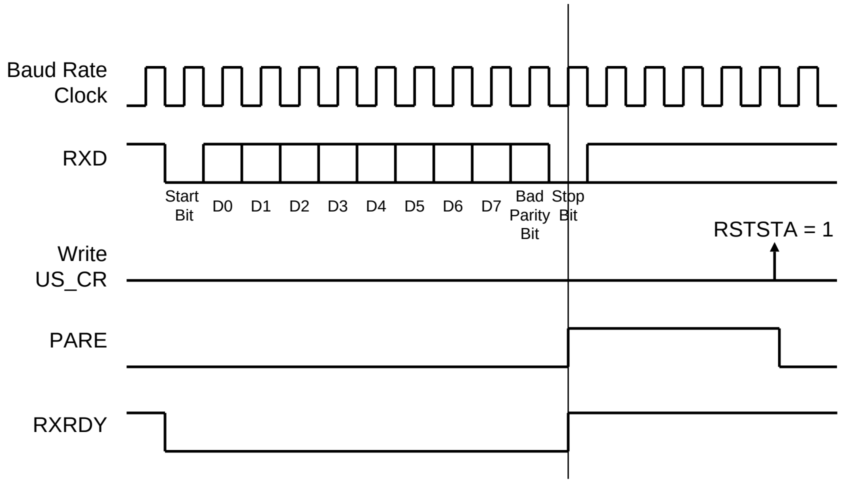
Table 26-6 shows an example of the parity bit for the character 0x41 (character ASCII "A") depending on the configuration of the USART. Because there are two bits at 1, 1 bit is added when a parity is odd, or 0 is added when a parity is even.

Table 26-6. Parity Bit Examples

Character	Hexa	Binary	Parity Bit	Parity Mode
A	0x41	0100 0001	1	Odd
A	0x41	0100 0001	0	Even
A	0x41	0100 0001	1	Mark
A	0x41	0100 0001	0	Space
A	0x41	0100 0001	None	None

When the receiver detects a parity error, it sets the PARE (Parity Error) bit in the Channel Status Register (US_CSR). The PARE bit can be cleared by writing the Control Register (US_CR) with the RSTSTA bit at 1. Figure 26-22 illustrates the parity bit status setting and clearing.

Figure 26-22. Parity Error



26.6.3.9 *Multidrop Mode*

If the PAR field in the Mode Register (US_MR) is programmed to the value 0x6 or 0x07, the USART runs in Multidrop Mode. This mode differentiates the data characters and the address characters. Data is transmitted with the parity bit at 0 and addresses are transmitted with the parity bit at 1.

If the USART is configured in multidrop mode, the receiver sets the PARE parity error bit when the parity bit is high and the transmitter is able to send a character with the parity bit high when the Control Register is written with the SENDA bit at 1.

To handle parity error, the PARE bit is cleared when the Control Register is written with the bit RSTSTA at 1.

The transmitter sends an address byte (parity bit set) when SENDA is written to US_CR. In this case, the next byte written to US_THR is transmitted as an address. Any character written in US_THR without having written the command SENDA is transmitted normally with the parity at 0.

26.6.3.10 *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 26-23, 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 26-23. Timeguard Operations

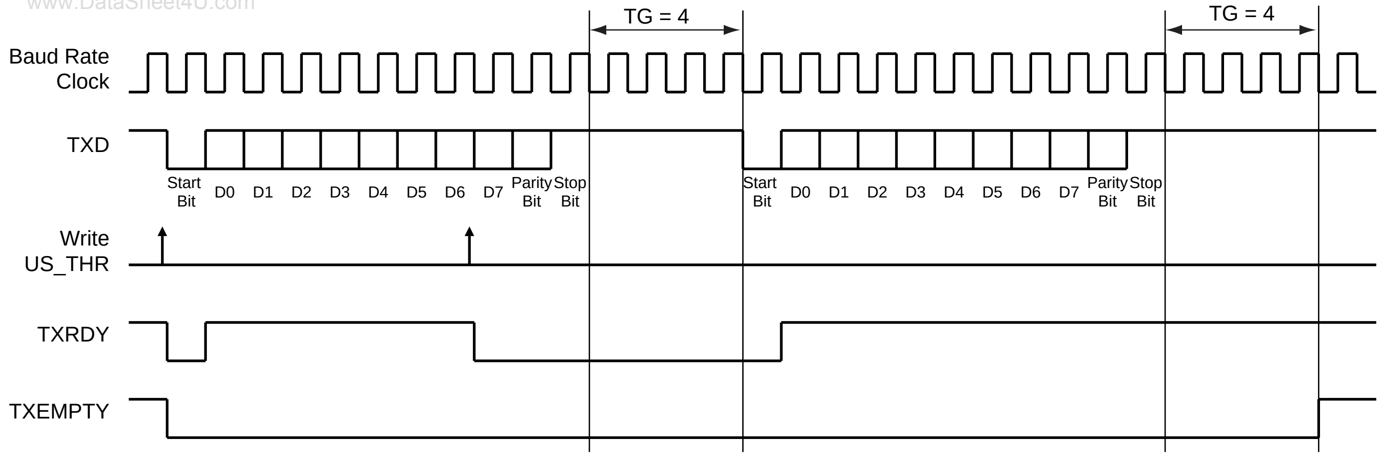


Table 26-7 indicates the maximum length of a timeguard period that the transmitter can handle in relation to the function of the Baud Rate.

Table 26-7. 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

26.6.3.11 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. Then, the user can either:

- Stop the counter clock until a new character is received. This is performed by writing the Control Register (US_CR) with the STTTO (Start Time-out) bit at 1. In this case, the idle state on RXD before a new character is received will not provide a time-out. This prevents having to handle an interrupt before a character is received and allows waiting for the next idle state on RXD after a frame is received.

- Obtain an interrupt while no character is received. This is performed by writing US_CR with the RETTO (Reload and Start Time-out) bit at 1. If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

If STTTO is performed, the counter clock is stopped until a first character is received. The idle state on RXD before the start of the frame does not provide a time-out. This prevents having to obtain a periodic interrupt and enables a wait of the end of frame when the idle state on RXD is detected.

If RETTO is performed, the counter starts counting down immediately from the value TO. This enables generation of a periodic interrupt so that a user time-out can be handled, for example when no key is pressed on a keyboard.

Figure 26-24 shows the block diagram of the Receiver Time-out feature.

Figure 26-24. Receiver Time-out Block Diagram

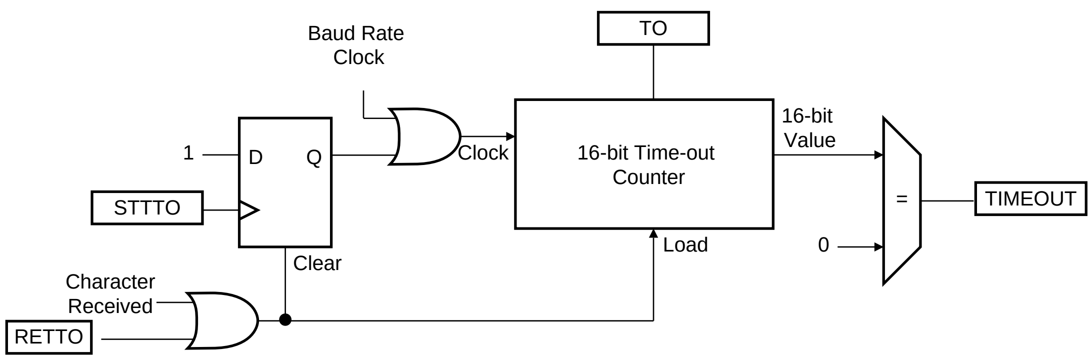


Table 26-8 gives the maximum time-out period for some standard baud rates.

Table 26-8. 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

Table 26-8. Maximum Time-out Period (Continued)

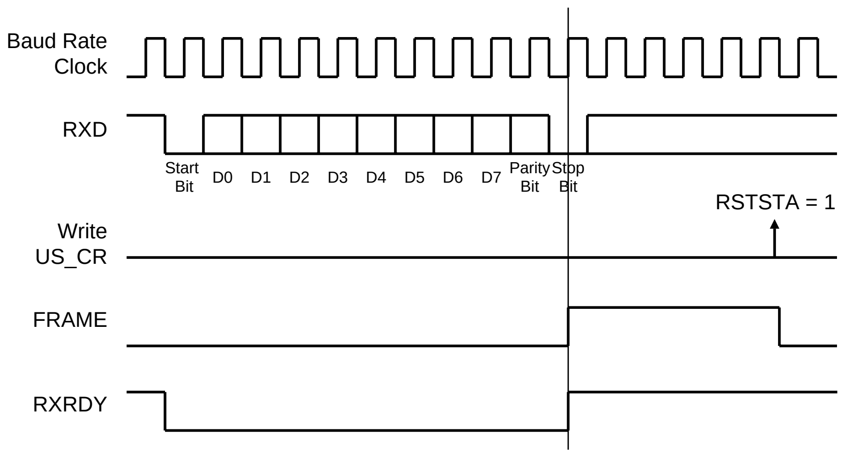
Baud Rate	Bit Time	Time-out
56000	18	1 170
57600	17	1 138
200000	5	328

26.6.3.12 *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 26-25. Framing Error Status



26.6.3.13 *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.

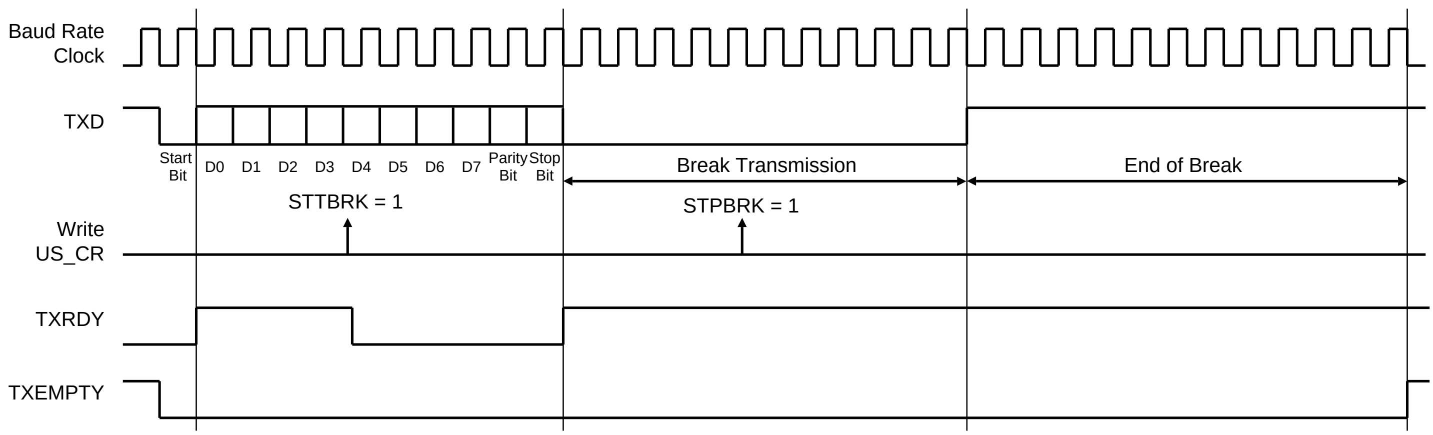
Writing US_CR with the both STTBRK and STPBRK bits at 1 can lead to an unpredictable result. All STPBRK commands requested without a previous STTBRK command are ignored. A byte written into the Transmit Holding Register while a break is pending, but not started, is ignored.

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 26-26 illustrates the effect of both the Start Break (STTBRK) and Stop Break (STPBRK) commands on the TXD line.

Figure 26-26. Break Transmission



26.6.3.14 *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.

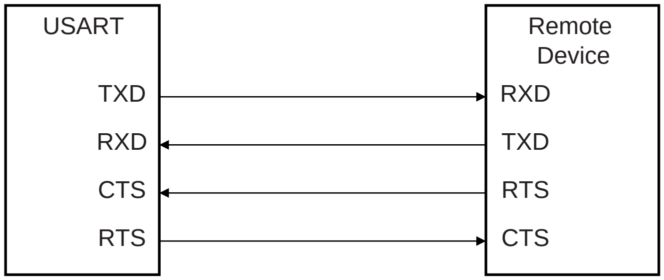
When the low stop bit is detected, the receiver asserts the RXBRK bit in US_CSR. This bit may be cleared by writing the Control Register (US_CR) with the bit RSTSTA at 1.

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.

26.6.3.15 *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 26-27.

Figure 26-27. 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 26-28 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 26-28. Receiver Behavior when Operating with Hardware Handshaking

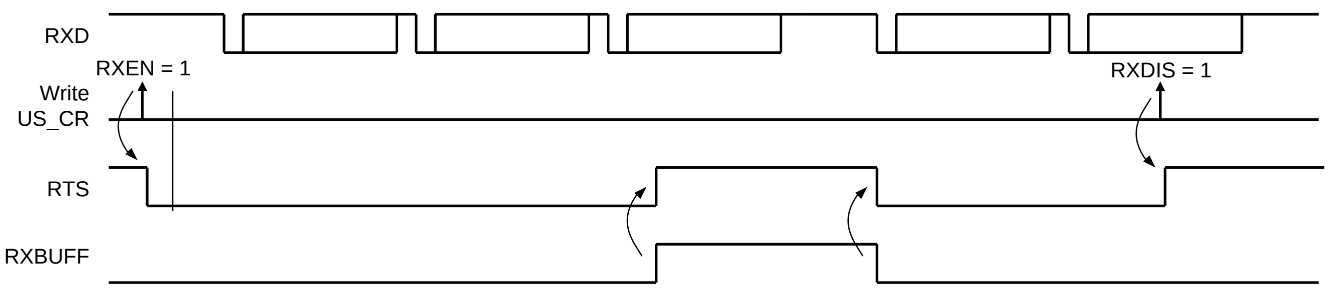
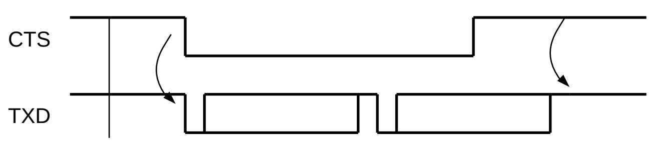


Figure 26-29 shows how the transmitter operates if hardware handshaking is enabled. The CTS pin disables the transmitter. If a character is being processing, the transmitter is disabled only after the completion of the current character and transmission of the next character happens as soon as the pin CTS falls.

Figure 26-29. Transmitter Behavior when Operating with Hardware Handshaking



26.6.4 ISO7816 Mode

The USART features an ISO7816-compatible operating mode. This mode permits interfacing with smart cards and Security Access Modules (SAM) communicating through an ISO7816 link. Both T = 0 and T = 1 protocols defined by the ISO7816 specification are supported.

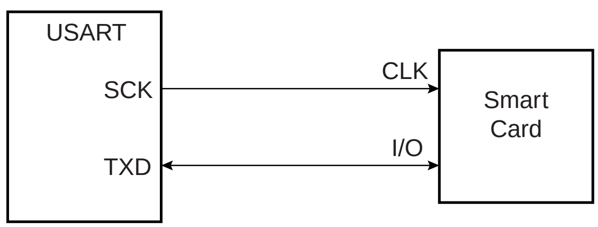
Setting the USART in ISO7816 mode is performed by writing the USART_MODE field in the Mode Register (US_MR) to the value 0x4 for protocol T = 0 and to the value 0x5 for protocol T = 1.

26.6.4.1 ISO7816 Mode Overview

The ISO7816 is a half duplex communication on only one bidirectional line. The baud rate is determined by a division of the clock provided to the remote device (see "Baud Rate Generator" on page 413).

The USART connects to a smart card as shown in Figure 26-30. 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 26-30. Connection of a Smart Card to the USART



When operating in ISO7816, either in T = 0 or T = 1 modes, the character format is fixed. The configuration is 8 data bits, even parity and 1 or 2 stop bits, regardless of the values programmed in the CHRL, MODE9, PAR and CHMODE fields. MSBF can be used to transmit LSB or MSB first. Parity Bit (PAR) can be used to transmit in normal or inverse mode. Refer to "USART Mode Register" on page 446 and "PAR: Parity Type" on page 447.

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).

26.6.4.2 Protocol T = 0

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

If no parity error is detected, the I/O line remains at 1 during the guard time and the transmitter can continue with the transmission of the next character, as shown in Figure 26-31.

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 26-32. 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 26-31. T = 0 Protocol without Parity Error

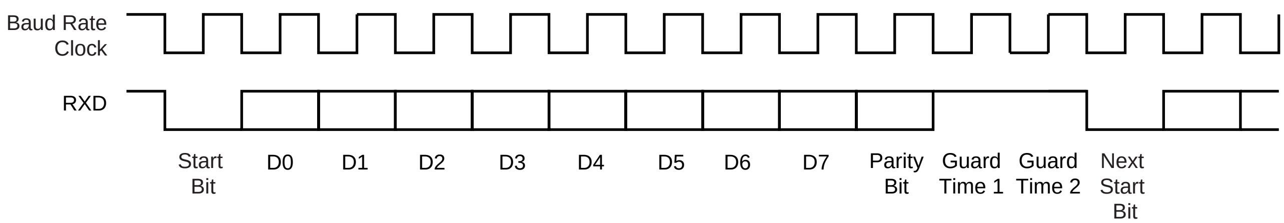
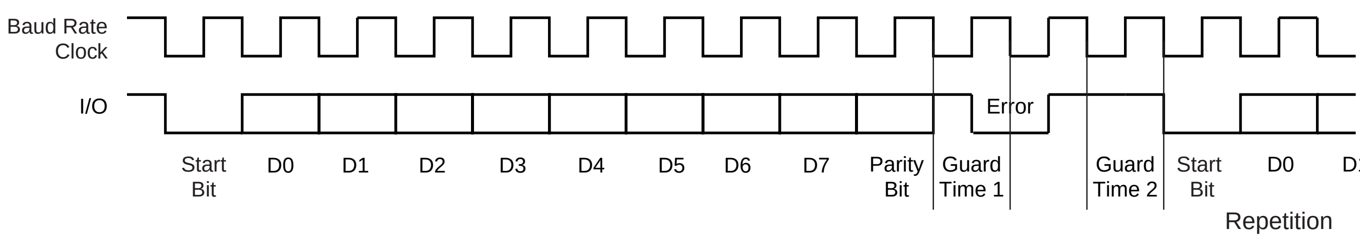


Figure 26-32. T = 0 Protocol with Parity Error



26.6.4.2.1 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.

26.6.4.2.2 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.

26.6.4.2.3 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.

26.6.4.2.4 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.

26.6.4.3 Protocol T = 1

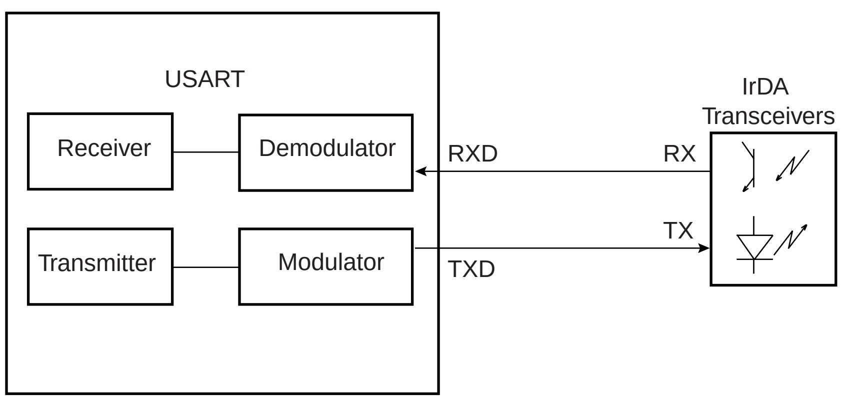
When operating in ISO7816 protocol T = 1, the transmission is similar to an asynchronous format with only one stop bit. The parity is generated when transmitting and checked when receiving. Parity error detection sets the PARE bit in the Channel Status Register (US_CSR).

26.6.5 IrDA Mode

The USART features an IrDA mode supplying half-duplex point-to-point wireless communication. It embeds the modulator and demodulator which allows a glueless connection to the infrared transceivers, as shown in Figure 26-33. The modulator and demodulator are compliant with the IrDA specification version 1.1 and support data transfer speeds ranging from 2.4 Kb/s to 115.2 Kb/s.

The USART IrDA mode is enabled by setting the USART_MODE field in the Mode Register (US_MR) to the value 0x8. The IrDA Filter Register (US_IF) allows configuring the demodulator filter. The USART transmitter and receiver operate in a normal asynchronous mode and all parameters are accessible. Note that the modulator and the demodulator are activated.

Figure 26-33. Connection to IrDA Transceivers



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

26.6.5.1 IrDA Modulation

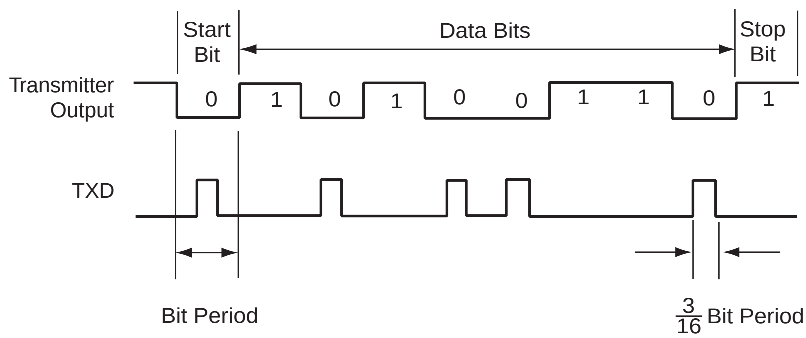
For baud rates up to and including 115.2 Kbits/sec, the RZI modulation scheme is used. "0" is represented by a light pulse of 3/16th of a bit time. Some examples of signal pulse duration are shown in Table 26-9.

Table 26-9. 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 26-34 shows an example of character transmission.

Figure 26-34. IrDA Modulation



26.6.5.2 IrDA Baud Rate

Table 26-10 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 26-10. IrDA Baud Rate Error

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

Table 26-10. IrDA Baud Rate Error (Continued)

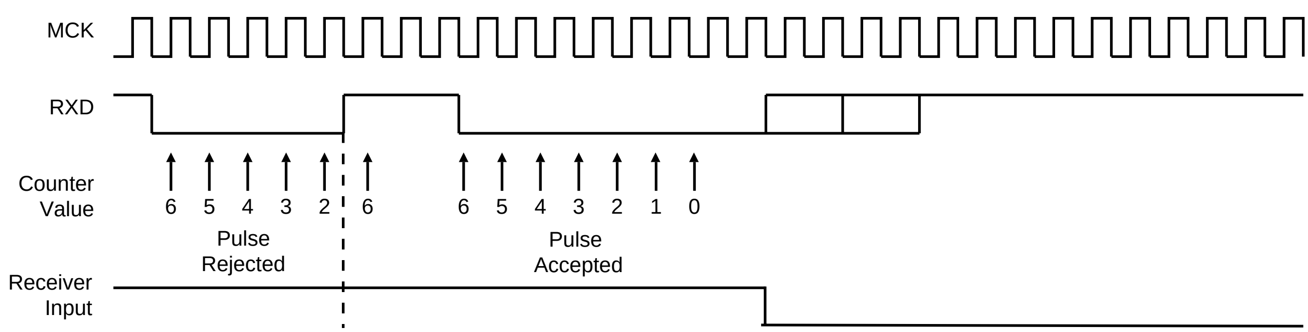
Peripheral Clock	Baud Rate	CD	Baud Rate Error	Pulse Time
32 768 000	38 400	53	0.63%	4.88
40 000 000	38 400	65	0.16%	4.88
3 686 400	19 200	12	0.00%	9.77
20 000 000	19 200	65	0.16%	9.77
32 768 000	19 200	107	0.31%	9.77
40 000 000	19 200	130	0.16%	9.77
3 686 400	9 600	24	0.00%	19.53
20 000 000	9 600	130	0.16%	19.53
32 768 000	9 600	213	0.16%	19.53
40 000 000	9 600	260	0.16%	19.53
3 686 400	2 400	96	0.00%	78.13
20 000 000	2 400	521	0.03%	78.13
32 768 000	2 400	853	0.04%	78.13

26.6.5.3 IrDA Demodulator

The demodulator is based on the IrDA Receive filter comprised of an 8-bit down counter which is loaded with the value programmed in US_IF. When a falling edge is detected on the RXD pin, the Filter Counter starts counting down at the 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 26-35 illustrates the operations of the IrDA demodulator.

Figure 26-35. IrDA Demodulator Operations

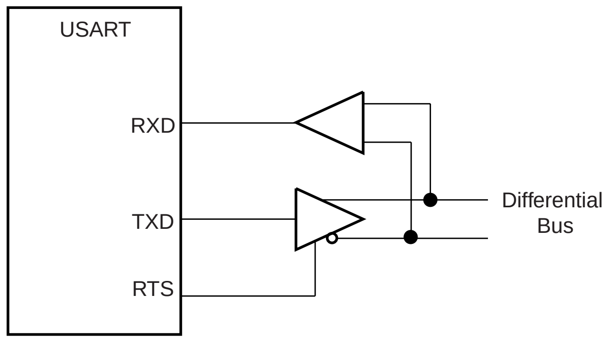


As the IrDA mode uses the same logic as the ISO7816, note that the FI_DI_RATIO field in US_FIDI must be set to a value higher than 0 in order to assure IrDA communications operate correctly.

26.6.6 RS485 Mode

The USART features the RS485 mode to enable line driver control. While operating in RS485 mode, the USART behaves as though in asynchronous or synchronous mode and configuration of all the parameters is possible. The difference is that the RTS pin is driven high when the transmitter is operating. The behavior of the RTS pin is controlled by the TXEMPTY bit. A typical connection of the USART to a RS485 bus is shown in Figure 26-36.

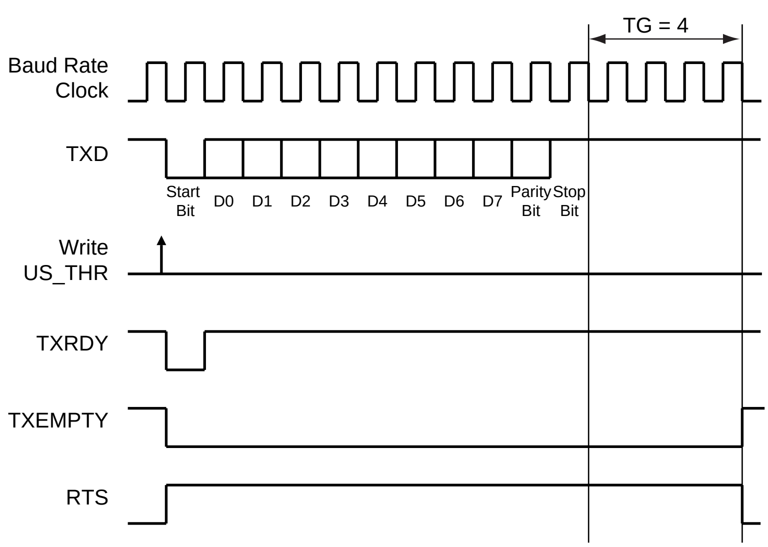
Figure 26-36. Typical Connection to a RS485 Bus



The USART is set in RS485 mode by programming the USART_MODE field in the Mode Register (US_MR) to the value 0x1.

The RTS pin is at a level inverse to the TXEMPTY bit. Significantly, the RTS pin remains high when a timeguard is programmed so that the line can remain driven after the last character completion. Figure 26-37 gives an example of the RTS waveform during a character transmission when the timeguard is enabled.

Figure 26-37. Example of RTS Drive with Timeguard



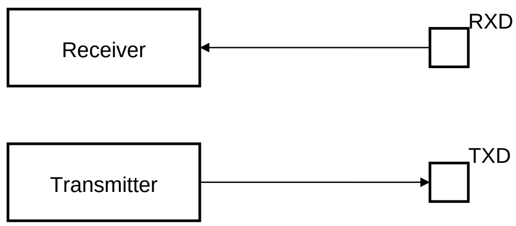
26.6.7 Test Modes

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.

26.6.7.1 Normal Mode

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

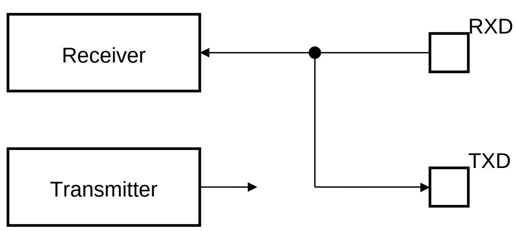
Figure 26-38. Normal Mode Configuration



26.6.7.2 Automatic Echo Mode

Automatic echo mode allows bit-by-bit retransmission. When a bit is received on the RXD pin, it is sent to the TXD pin, as shown in Figure 26-39. 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.

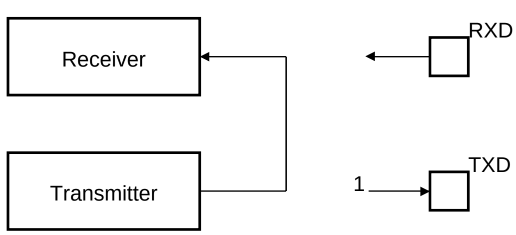
Figure 26-39. Automatic Echo Mode Configuration



26.6.7.3 Local Loopback Mode

Local loopback mode connects the output of the transmitter directly to the input of the receiver, as shown in Figure 26-40. 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.

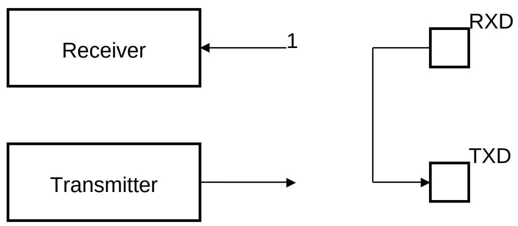
Figure 26-40. Local Loopback Mode Configuration



26.6.7.4 *Remote Loopback Mode*

Remote loopback mode directly connects the RXD pin to the TXD pin, as shown in Figure 26-41. The transmitter and the receiver are disabled and have no effect. This mode allows bit-by-bit retransmission.

Figure 26-41. Remote Loopback Mode Configuration



26.7 USART User Interface

Table 26-11. USART Memory Map

Offset	Register	Name	Access	Reset State
0x0000	Control Register	US_CR	Write-only	–
0x0004	Mode Register	US_MR	Read/Write	–
0x0008	Interrupt Enable Register	US_IER	Write-only	–
0x000C	Interrupt Disable Register	US_IDR	Write-only	–
0x0010	Interrupt Mask Register	US_IMR	Read-only	0x0
0x0014	Channel Status Register	US_CSR	Read-only	–
0x0018	Receiver Holding Register	US_RHR	Read-only	0x0
0x001C	Transmitter Holding Register	US_THR	Write-only	–
0x0020	Baud Rate Generator Register	US_BRGR	Read/Write	0x0
0x0024	Receiver Time-out Register	US_RTOR	Read/Write	0x0
0x0028	Transmitter Timeguard Register	US_TTGR	Read/Write	0x0
0x2C - 0x3C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read/Write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	–
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read/Write	0x0
0x0050	Manchester Encoder Decoder Register	US_MAN	Read/Write	0x30011004
0x5C - 0xFC	Reserved	–	–	–
0x100 - 0x128	Reserved for PDC Registers	–	–	–

26.7.1 USART Control Register

Name: US_CR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
RETTO	RSTNACK	RSTIT	SENDA	STTTO	STPBRK	STTBRK	RSTSTA

7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

- **RSTRX: Reset Receiver**

0: No effect.

1: Resets the receiver.

- **RSTTX: Reset Transmitter**

0: No effect.

1: Resets the transmitter.

- **RXEN: Receiver Enable**

0: No effect.

1: Enables the receiver, if RXDIS is 0.

- **RXDIS: Receiver Disable**

0: No effect.

1: Disables the receiver.

- **TXEN: Transmitter Enable**

0: No effect.

1: Enables the transmitter if TXDIS is 0.

- **TXDIS: Transmitter Disable**

0: No effect.

1: Disables the transmitter.

- **RSTSTA: Reset Status Bits**

0: No effect.

1: Resets the status bits PARE, FRAME, OVRE, MANERR and RXBRK in US_CSR.

- **STTBRK: Start Break**

0: No effect.

1: Starts transmission of a break after the characters present in US_THR and the Transmit Shift Register have been transmitted. No effect if a break is already being transmitted.

- **STPBRK: Stop Break**

0: No effect.

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods. No effect if no break is being transmitted.

- **STTTO: Start Time-out**

0: No effect.

1: Starts waiting for a character before clocking the time-out counter. Resets the status bit TIMEOUT in US_CSR.

- **SENDA: Send Address**

0: No effect.

1: In Multidrop Mode only, the next character written to the US_THR is sent with the address bit set.

- **RSTIT: Reset Iterations**

0: No effect.

1: Resets ITERATION in US_CSR. No effect if the ISO7816 is not enabled.

- **RSTNACK: Reset Non Acknowledge**

0: No effect

1: Resets NACK in US_CSR.

- **RETTO: Rearm Time-out**

0: No effect

1: Restart Time-out

- **RTSEN: Request to Send Enable**

0: No effect.

1: Drives the pin RTS to 0.

- **RTSDIS: Request to Send Disable**

0: No effect.

1: Drives the pin RTS to 1.

26.7.2 USART Mode Register

Name: US_MR

Access Type: Read/Write

31	30	29	28	27	26	25	24
ONEBIT	MODSYNC–	MAN	FILTER	–	MAX_ITERATION		

23	22	21	20	19	18	17	16
–	VAR_SYNC	DSNACK	INACK	OVER	CLKO	MODE9	MSBF

15	14	13	12	11	10	9	8
CHMODE		NBSTOP		PAR			SYNC

7	6	5	4	3	2	1	0
CHRL		USCLKS		USART_MODE			

- **USART_MODE**

USART_MODE				Mode of the USART
0	0	0	0	Normal
0	0	0	1	RS485
0	0	1	0	Hardware Handshaking
0	0	1	1	Reserved
0	1	0	0	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 (DIV = 8)
1	0	Reserved
1	1	SCK

- **CHRL: Character Length.**

CHRL		Character Length
0	0	5 bits

0	1	6 bits
1	0	7 bits
1	1	8 bits

- **SYNC: Synchronous Mode Select**

0: USART operates in Asynchronous Mode.

1: USART operates in Synchronous Mode.

- **PAR: Parity Type**

PAR			Parity Type
0	0	0	Even parity
0	0	1	Odd parity
0	1	0	Parity forced to 0 (Space)
0	1	1	Parity forced to 1 (Mark)
1	0	x	No parity
1	1	x	Multidrop mode

- **NBSTOP: Number of Stop Bits**

NBSTOP		Asynchronous (SYNC = 0)	Synchronous (SYNC = 1)
0	0	1 stop bit	1 stop bit
0	1	1.5 stop bits	Reserved
1	0	2 stop bits	2 stop bits
1	1	Reserved	Reserved

- **CHMODE: Channel Mode**

CHMODE		Mode Description
0	0	Normal Mode
0	1	Automatic Echo. Receiver input is connected to the TXD pin.
1	0	Local Loopback. Transmitter output is connected to the Receiver Input..
1	1	Remote Loopback. RXD pin is internally connected to the TXD pin.

- **MSBF: Bit Order**

0: Least Significant Bit is sent/received first.

1: Most Significant Bit is sent/received first.

- **MODE9: 9-bit Character Length**

0: CHRL defines character length.

1: 9-bit character length.

- **CLKO: Clock Output Select**

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

- **OVER: Oversampling Mode**

0: 16x Oversampling.

1: 8x Oversampling.

- **INACK: Inhibit Non Acknowledge**

0: The NACK is generated.

1: The NACK is not generated.

- **DSNACK: Disable Successive NACK**

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.

- **VAR_SYNC: Variable Synchronization of Command/Data Sync Start Frame Delimiter**

0: User defined configuration of command or data sync field depending on SYNC value.

1: The sync field is updated when a character is written into US_THR register.

- **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).

- **MAN: Manchester Encoder/Decoder Enable**

0: Manchester Encoder/Decoder are disabled.

1: Manchester Encoder/Decoder are enabled.

- **MODSYNC: Manchester Synchronization Mode**

0:The Manchester Start bit is a 0 to 1 transition

1: The Manchester Start bit is a 1 to 0 transition.

- **ONEBIT: Start Frame Delimiter Selector**

0: Start Frame delimiter is COMMAND or DATA SYNC.

1: Start Frame delimiter is One Bit.

26.7.3 USART Interrupt Enable Register

Name: US_IER

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITERATION	TXEMPTY	TIMEOUT

7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **RXRDY: RXRDY Interrupt Enable**
- **TXRDY: TXRDY Interrupt Enable**
- **RXBRK: Receiver Break Interrupt Enable**
- **ENDRX: End of Receive Transfer Interrupt Enable**
- **ENDTX: End of Transmit Interrupt Enable**
- **OVRE: Overrun Error Interrupt Enable**
- **FRAME: Framing Error Interrupt Enable**
- **PARE: Parity Error Interrupt Enable**
- **TIMEOUT: Time-out Interrupt Enable**
- **TXEMPTY: TXEMPTY Interrupt Enable**
- **ITERATION: Iteration Interrupt Enable**
- **TXBUFE: Buffer Empty Interrupt Enable**
- **RXBUFF: Buffer Full Interrupt Enable**
- **NACK: Non Acknowledge Interrupt Enable**
- **CTSIC: Clear to Send Input Change Interrupt Enable**
- **MANE: Manchester Error Interrupt Enable**

0: No effect.

1: Enables the corresponding interrupt.

26.7.4 USART Interrupt Disable Register

Name: US_IDR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITERATION	TXEMPTY	TIMEOUT

7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **RXRDY: RXRDY Interrupt Disable**
- **TXRDY: TXRDY Interrupt Disable**
- **RXBRK: Receiver Break Interrupt Disable**
- **ENDRX: End of Receive Transfer Interrupt Disable**
- **ENDTX: End of Transmit Interrupt Disable**
- **OVRE: Overrun Error Interrupt Disable**
- **FRAME: Framing Error Interrupt Disable**
- **PARE: Parity Error Interrupt Disable**
- **TIMEOUT: Time-out Interrupt Disable**
- **TXEMPTY: TXEMPTY Interrupt Disable**
- **ITERATION: Iteration Interrupt Disable**
- **TXBUFE: Buffer Empty Interrupt Disable**
- **RXBUFF: Buffer Full Interrupt Disable**
- **NACK: Non Acknowledge Interrupt Disable**
- **CTSIC: Clear to Send Input Change Interrupt Disable**
- **MANE: Manchester Error Interrupt Disable**

0: No effect.

1: Disables the corresponding interrupt.

26.7.5 USART Interrupt Mask Register

Name: US_IMR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITERATION	TXEMPTY	TIMEOUT

7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **RXRDY: RXRDY Interrupt Mask**
- **TXRDY: TXRDY Interrupt Mask**
- **RXBRK: Receiver Break Interrupt Mask**
- **ENDRX: End of Receive Transfer Interrupt Mask**
- **ENDTX: End of Transmit Interrupt Mask**
- **OVRE: Overrun Error Interrupt Mask**
- **FRAME: Framing Error Interrupt Mask**
- **PARE: Parity Error Interrupt Mask**
- **TIMEOUT: Time-out Interrupt Mask**
- **TXEMPTY: TXEMPTY Interrupt Mask**
- **ITERATION: Iteration Interrupt Mask**
- **TXBUFE: Buffer Empty Interrupt Mask**
- **RXBUFF: Buffer Full Interrupt Mask**
- **NACK: Non Acknowledge Interrupt Mask**
- **CTSIC: Clear to Send Input Change Interrupt Mask**
- **MANE: Manchester Error Interrupt Mask**

0: The corresponding interrupt is disabled.

1: The corresponding interrupt is enabled.

26.7.6 USART Channel Status Register

Name: US_CSR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITERATION	TXEMPTY	TIMEOUT

7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

- **RXRDY: Receiver Ready**

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

- **TXRDY: Transmitter Ready**

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

- **RXBRK: Break Received/End of Break**

0: No Break received or End of Break detected since the last RSTSTA.

1: Break Received or End of Break detected since the last RSTSTA.

- **ENDRX: End of Receiver Transfer**

0: The End of Transfer signal from the Receive PDC channel is inactive.

1: The End of Transfer signal from the Receive PDC channel is active.

- **ENDTX: End of Transmitter Transfer**

0: The End of Transfer signal from the Transmit PDC channel is inactive.

1: The End of Transfer signal from the Transmit PDC channel is active.

- **OVRE: Overrun Error**

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

- **FRAME: Framing Error**

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

- **PARE: Parity Error**

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

- **TIMEOUT: Receiver Time-out**

0: There has not been a time-out since the last Start Time-out command (STTTO in US_CR) or the Time-out Register is 0.

1: There has been a time-out since the last Start Time-out command (STTTO in US_CR).

- **TXEMPTY: Transmitter Empty**

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

- **ITERATION: Max number of Repetitions Reached**

0: Maximum number of repetitions has not been reached since the last RSIT.

1: Maximum number of repetitions has been reached since the last RSIT.

- **TXBUFE: Transmission Buffer Empty**

0: The signal Buffer Empty from the Transmit PDC channel is inactive.

1: The signal Buffer Empty from the Transmit PDC channel is active.

- **RXBUFF: Reception Buffer Full**

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

- **NACK: Non Acknowledge**

0: No Non Acknowledge has not been detected since the last RSTNACK.

1: At least one Non Acknowledge has been detected since the last RSTNACK.

- **CTSIC: Clear to Send Input Change Flag**

0: No input change has been detected on the CTS pin since the last read of US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US_CSR.

- **CTS: Image of CTS Input**

0: CTS is at 0.

1: CTS is at 1.

- **MANERR: Manchester Error**

0: No Manchester error has been detected since the last RSTSTA.

1: At least one Manchester error has been detected since the last RSTSTA.

26.7.7 USART Receive Holding Register

Name: US_RHR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
RXSYNH	–	–	–	–	–	–	RXCHR

7	6	5	4	3	2	1	0
RXCHR							

- **RXCHR: Received Character**

Last character received if RXRDY is set.

- **RXSYNH: Received Sync**

0: Last Character received is a Data.

1: Last Character received is a Command.

26.7.8 USART Transmit Holding Register

Name: US_THR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
TXSYNH	–	–	–	–	–	–	TXCHR

7	6	5	4	3	2	1	0
TXCHR							

- **TXCHR: Character to be Transmitted**

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

- **TXSYNH: Sync Field to be transmitted**

0: The next character sent is encoded as a data. Start Frame Delimiter is DATA SYNC.

1: The next character sent is encoded as a command. Start Frame Delimiter is COMMAND SYNC.

26.7.9 USART Baud Rate Generator Register

Name: US_BRGR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
CD							

7	6	5	4	3	2	1	0
CD							

- **CD: Clock Divider**

<table>
<tr><td rowspan="3">CD</td><td colspan="3">USART_MODE ≠ ISO7816</td><td rowspan="3">USART_MODE = ISO7816</td></tr>
<tr><td colspan="2">SYNC = 0</td><td>SYNC = 1</td></tr>
<tr><td>OVER = 0</td><td>OVER = 1</td><td></td></tr>
<tr><td>0</td><td colspan="4">Baud Rate Clock Disabled</td></tr>
<tr><td>1 to 65535</td><td>Baud Rate = Selected Clock/16/CD</td><td>Baud Rate = Selected Clock/8/CD</td><td>Baud Rate = Selected Clock /CD</td><td>Baud Rate = Selected Clock/CD/FI_DI_RATIO</td></tr>
</table>

- **FP: Fractional Part**

0: Fractional divider is disabled.

1 - 7: Baudrate resolution, defined by FP x 1/8.

26.7.10 USART Receiver Time-out Register

Name: US_RTOR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
TO							

7	6	5	4	3	2	1	0
TO							

- **TO: Time-out Value**

0: The Receiver Time-out is disabled.

1 - 65535: The Receiver Time-out is enabled and the Time-out delay is TO x Bit Period.

26.7.11 USART Transmitter Timeguard Register

Name: US_TTGR

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
TG							

- **TG: Timeguard Value**

0: The Transmitter Timeguard is disabled.

1 - 255: The Transmitter timeguard is enabled and the timeguard delay is TG x Bit Period.

26.7.12 USART FI DI RATIO Register

Name: US_FIDI

Access Type: Read/Write

Reset Value : 0x174

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

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

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

7	6	5	4	3	2	1	0
FI_DI_RATIO							

- **FI_DI_RATIO: FI Over DI Ratio Value**

0: If ISO7816 mode is selected, the Baud Rate Generator generates no signal.

1 - 2047: If ISO7816 mode is selected, the Baud Rate is the clock provided on SCK divided by FI_DI_RATIO.

26.7.13 USART Number of Errors Register

Name: US_NER

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
NB_ERRORS							

- **NB_ERRORS: Number of Errors**

Total number of errors that occurred during an ISO7816 transfer. This register automatically clears when read.

26.7.14 USART Manchester Configuration Register

Name: US_MAN

Access Type: Read/Write

31	30	29	28	27	26	25	24
–	DRIFT	–	RX_MPOL	–	–	RX_PP	

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

15	14	13	12	11	10	9	8
–	–	–	TX_MPOL	–	–	TX_PP	

7	6	5	4	3	2	1	0
–	–	–	–	TX_PL			

- **TX_PL: Transmitter Preamble Length**

0: The Transmitter Preamble pattern generation is disabled

1 - 15: The Preamble Length is TX_PL x Bit Period

- **TX_PP: Transmitter Preamble Pattern**

TX_PP		Preamble Pattern default polarity assumed (TX_MPOL field not set)
0	0	ALL_ONE
0	1	ALL_ZERO
1	0	ZERO_ONE
1	1	ONE_ZERO

- **TX_MPOL: Transmitter Manchester Polarity**

0: Logic Zero is coded as a zero-to-one transition, Logic One is coded as a one-to-zero transition.

1: Logic Zero is coded as a one-to-zero transition, Logic One is coded as a zero-to-one transition.

- **RX_PL: Receiver Preamble Length**

0: The receiver preamble pattern detection is disabled

1 - 15: The detected preamble length is RX_PL x Bit Period

- **RX_PP: Receiver Preamble Pattern detected**

RX_PP		Preamble Pattern default polarity assumed (RX_MPOL field not set)
0	0	ALL_ONE
0	1	ALL_ZERO
1	0	ZERO_ONE
1	1	ONE_ZERO

- **RX_MPOL: Receiver Manchester Polarity**

0: Logic Zero is coded as a zero-to-one transition, Logic One is coded as a one-to-zero transition.

1: Logic Zero is coded as a one-to-zero transition, Logic One is coded as a zero-to-one transition.

- **DRIFT: Drift compensation**

0: The USART can not recover from an important clock drift

1: The USART can recover from clock drift. The 16X clock mode must be enabled.

26.7.15 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.

27. Serial Synchronous Controller (SSC)

27.1 Scope Description

The Atmel Synchronous Serial Controller (SSC) provides a synchronous communication link with external devices. It supports many serial synchronous communication protocols generally used in audio and telecom applications such as I2S, Short Frame Sync, Long Frame Sync, etc.

The SSC contains an independent receiver and transmitter and a common clock divider. The receiver and the transmitter each interface with three signals: the TD/RD signal for data, the TK/RK signal for the clock and the TF/RF signal for the Frame Sync. The transfers can be programmed to start automatically or on different events detected on the Frame Sync signal.

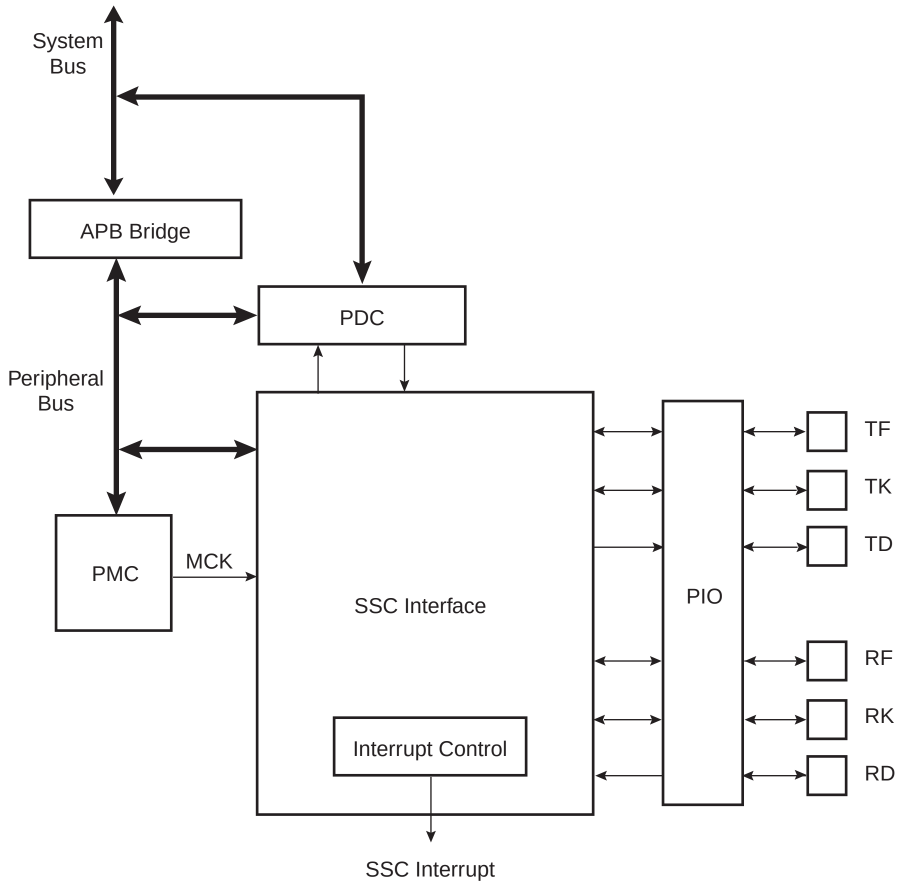
The SSC's high-level of programmability and its two dedicated PDC channels of up to 32 bits permit a continuous high bit rate data transfer without processor intervention.

Featuring connection to two PDC channels, the SSC permits interfacing with low processor overhead to the following:

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

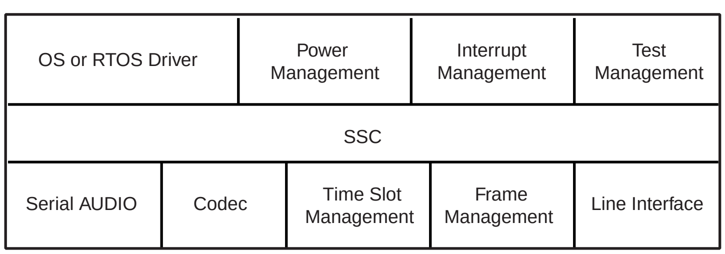
27.2 Block Diagram

Figure 27-1. Block Diagram



27.3 Application Block Diagram

Figure 27-2. Application Block Diagram



27.4 Pin Name List

Table 27-1. I/O Lines Description

Pin Name	Pin Description	Type
RF	Receiver Frame Synchro	Input/Output
RK	Receiver Clock	Input/Output
RD	Receiver Data	Input
TF	Transmitter Frame Synchro	Input/Output
TK	Transmitter Clock	Input/Output
TD	Transmitter Data	Output

27.5 Product Dependencies

27.5.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines.

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.

27.5.2 Power Management

The SSC is not continuously clocked. The SSC interface may be clocked through the Power Management Controller (PMC), therefore the programmer must first configure the PMC to enable the SSC clock.

27.5.3 Interrupt

The SSC interface has an interrupt line connected to the Advanced Interrupt Controller (AIC). Handling interrupts requires programming the AIC before configuring the SSC.

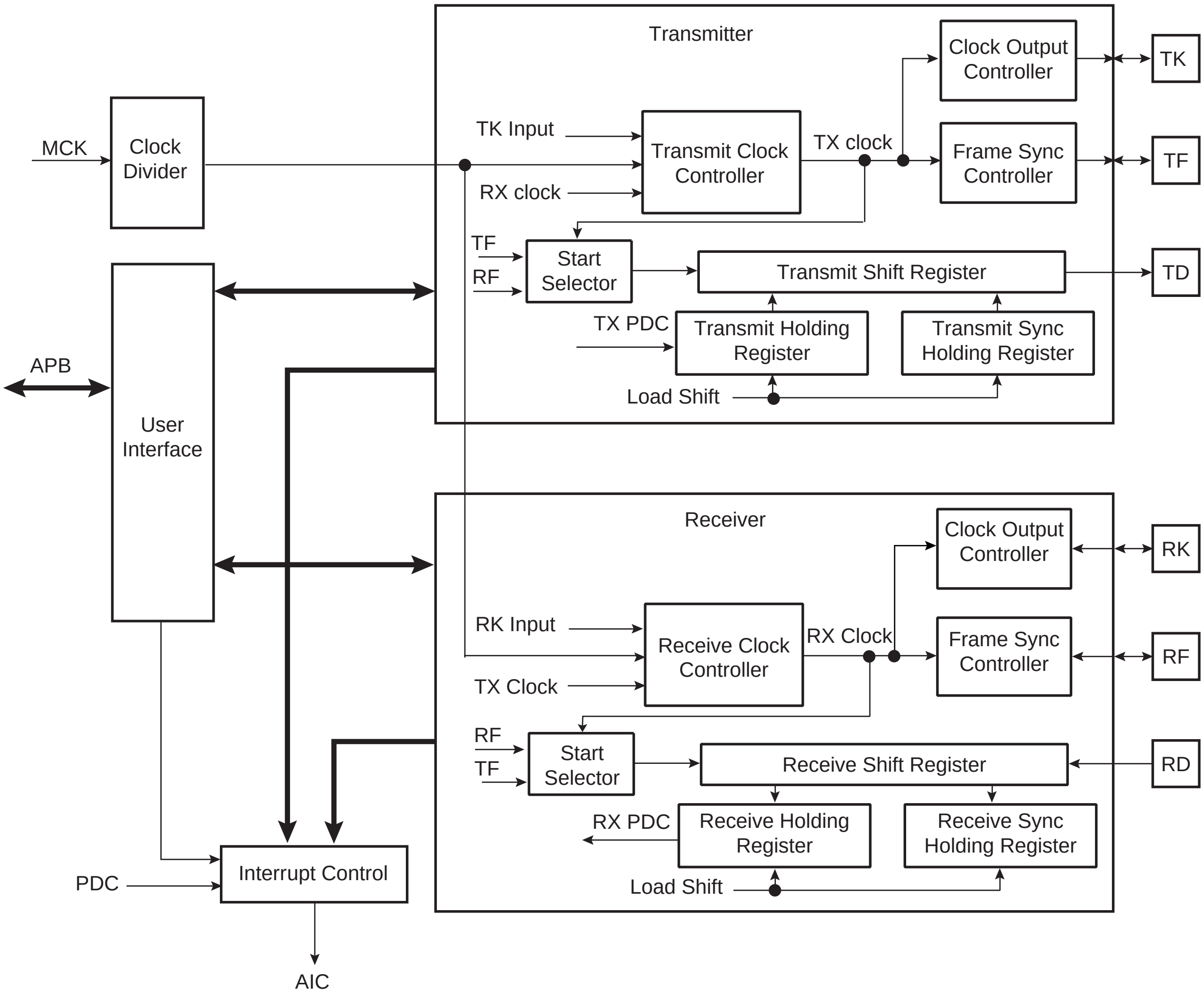
All SSC interrupts can be enabled/disabled configuring the SSC Interrupt mask register. Each pending and unmasked SSC interrupt will assert the SSC interrupt line. The SSC interrupt service routine can get the interrupt origin by reading the SSC interrupt status register.

27.6 Functional Description

This chapter contains the functional description of the following: SSC Functional Block, Clock Management, Data format, Start, Transmitter, Receiver and Frame Sync.

The receiver and transmitter operate separately. However, they can work synchronously by programming the receiver to use the transmit clock and/or to start a data transfer when transmission starts. Alternatively, this can be done by programming the transmitter to use the receive clock and/or to start a data transfer when reception starts. The transmitter and the receiver can be programmed to operate with the clock signals provided on either the TK or RK pins. This allows the SSC to support many slave-mode data transfers. The maximum clock speed allowed on the TK and RK pins is the master clock divided by 2.

Figure 27-3. SSC Functional Block Diagram



27.6.1 Clock Management

The transmitter clock can be generated by:

- an external clock received on the TK I/O pad
- the receiver clock
- the internal clock divider

The receiver clock can be generated by:

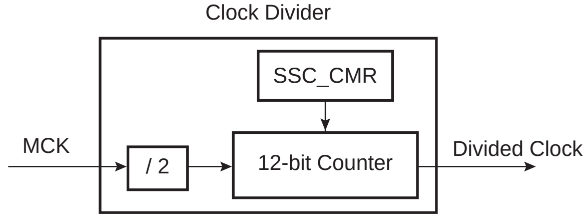
- an external clock received on the RK I/O pad
- the transmitter clock
- the internal clock divider

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

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

27.6.1.1 Clock Divider

Figure 27-4. Divided Clock Block Diagram



The Master Clock divider is determined by the 12-bit field DIV counter and comparator (so its maximal value is 4095) in the Clock Mode Register SSC_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 to or greater than 1, the Divided Clock has a frequency of Master Clock divided by 2 times DIV. Each level of the Divided Clock has a duration of the Master Clock multiplied by DIV. This ensures a 50% duty cycle for the Divided Clock regardless of whether the DIV value is even or odd.

Figure 27-5. Divided Clock Generation

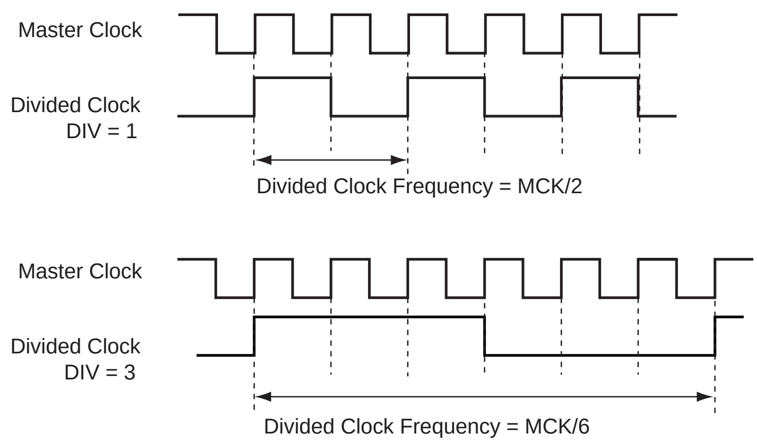


Table 27-2.

Maximum	Minimum
MCK / 2	MCK / 8190

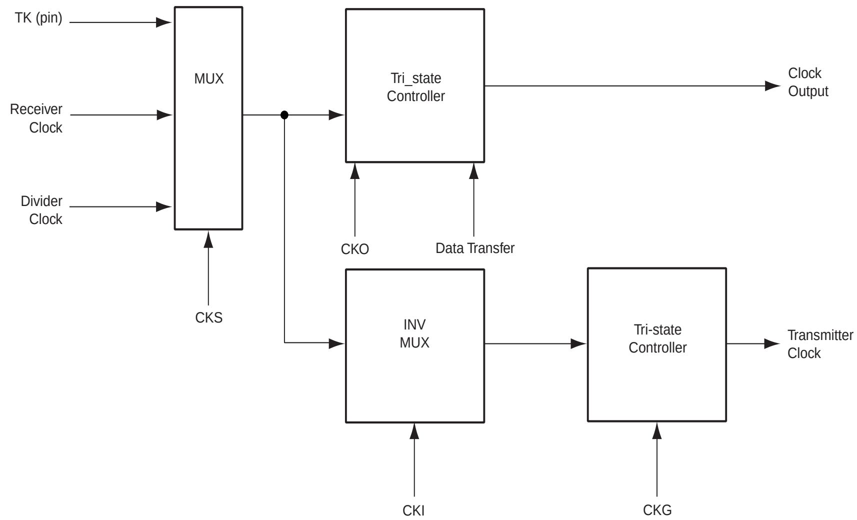
27.6.1.2 Transmitter Clock Management

The transmitter clock is generated from the receiver clock or the divider clock or an external clock scanned on the TK I/O pad. The transmitter clock is selected by the CKS field in SSC_TCMR (Transmit Clock Mode Register). Transmit Clock can be inverted independently by the CKI bits in SSC_TCMR.

The transmitter can also drive the TK I/O pad continuously or be limited to the actual data transfer. The clock output is configured by the SSC_TCMR register. The Transmit Clock 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 27-6. Transmitter Clock Management

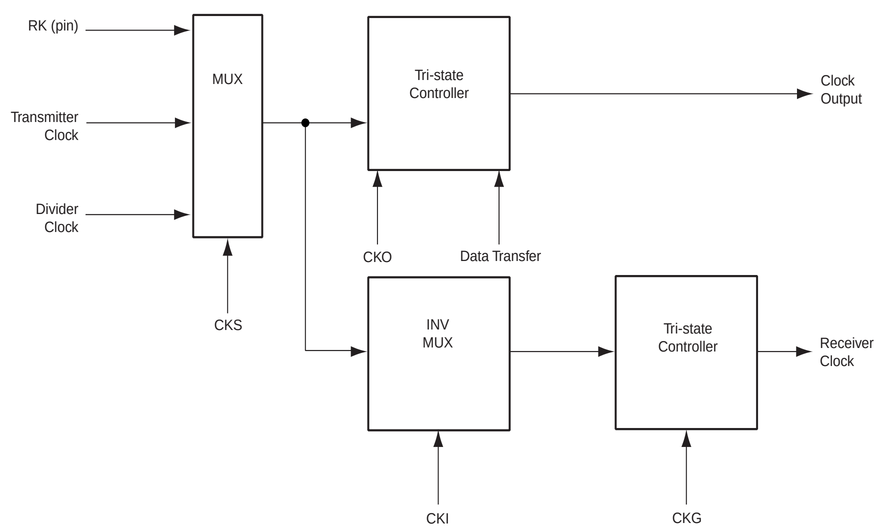


27.6.1.3 Receiver Clock Management

The receiver clock is generated from the transmitter clock or the divider clock or an external clock scanned on the RK I/O pad. The Receive Clock is selected by the CKS field in SSC_RCMR (Receive Clock Mode Register). Receive Clocks can be inverted independently by the CKI bits in SSC_RCMR.

The receiver can also drive the RK I/O pad continuously or be limited to the actual data transfer. The clock output is configured by the SSC_RCMR register. The Receive Clock Inversion (CKI) bits have no effect on the clock outputs. Programming the RCMR register to select RK pin (CKS field) and at the same time Continuous Receive Clock (CKO field) can lead to unpredictable results.

Figure 27-7. Receiver Clock Management



27.6.1.4 Serial Clock Ratio Considerations

The Transmitter and the Receiver can be programmed to operate with the clock signals provided on either the TK or RK pins. This allows the SSC to support many slave-mode data transfers. In this case, the maximum clock speed allowed on the RK pin is:

– Master Clock divided by 2 if Receiver Frame Synchro is input

– Master Clock divided by 3 if Receiver Frame Synchro is output

In addition, the maximum clock speed allowed on the TK pin is:

– Master Clock divided by 6 if Transmit Frame Synchro is input

– Master Clock divided by 2 if Transmit Frame Synchro is output

27.6.2 Transmitter Operations

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

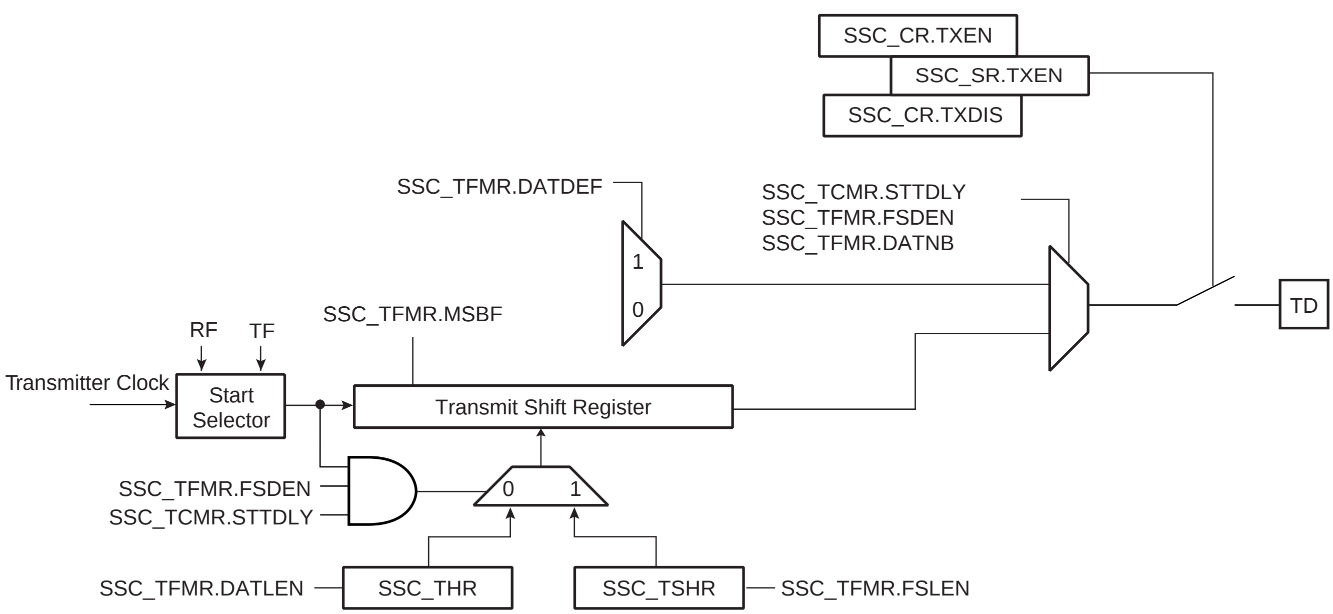
The start event is configured by setting the Transmit Clock Mode Register (SSC_TCMR). See "Start" on page 470.

The frame synchronization is configured setting the Transmit Frame Mode Register (SSC_TFMR). See "Frame Sync" on page 472.

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 27-8. Transmitter Block Diagram



27.6.3 Receiver Operations

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

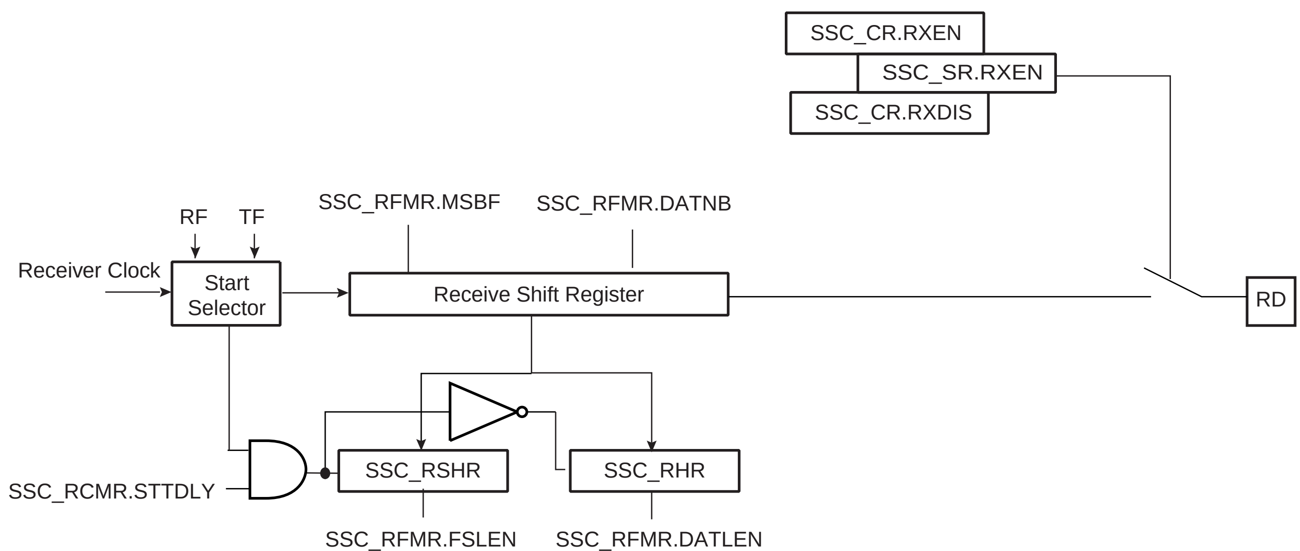
The start event is configured setting the Receive Clock Mode Register (SSC_RCMR). See "Start" on page 470.

The frame synchronization is configured setting the Receive Frame Mode Register (SSC_RFMR). See "Frame Sync" on page 472.

The receiver uses a shift register clocked by the receiver clock signal and the start mode selected in the SSC_RCMR. The data is transferred from the shift register depending on the data format selected.

When the receiver shift register is full, the SSC transfers this data in the holding register, the status flag RXRDY is set in SSC_SR and the data can be read in the receiver holding register. If another transfer occurs before read of the RHR register, the status flag OVERUN is set in SSC_SR and the receiver shift register is transferred in the RHR register.

Figure 27-9. Receiver Block Diagram



27.6.4 Start

The transmitter and receiver can both be programmed to start their operations when an event occurs, respectively in the Transmit Start Selection (START) field of SSC_TCMR and in the Receive Start Selection (START) field of SSC_RCMR.

Under the following conditions the start event is independently programmable:

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

A start can be programmed in the same manner on either side of the Transmit/Receive Clock Register (RCMR/TCMR). Thus, the start could be on TF (Transmit) or RF (Receive).

Moreover, the Receiver can start when data is detected in the bit stream with the Compare Functions.

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

Figure 27-10. Transmit Start Mode

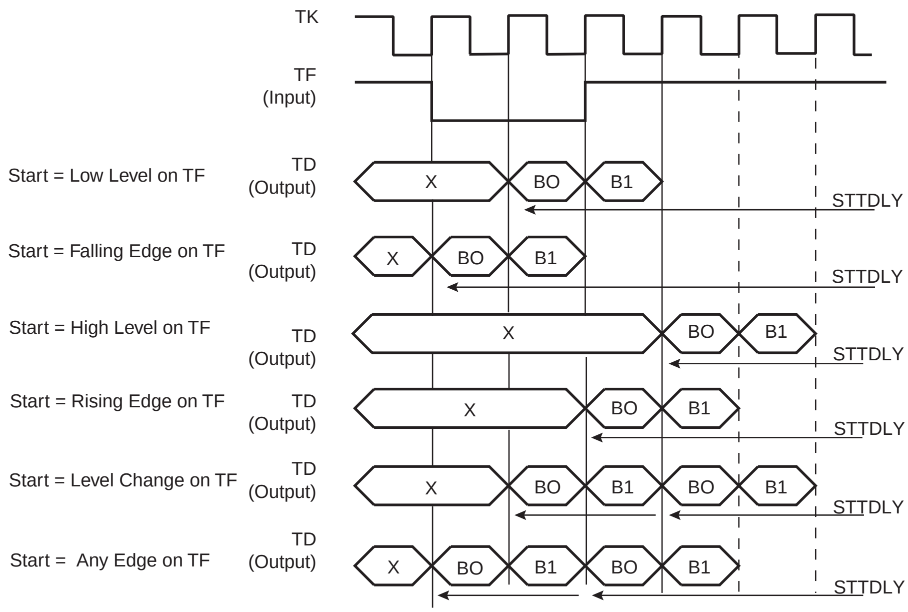
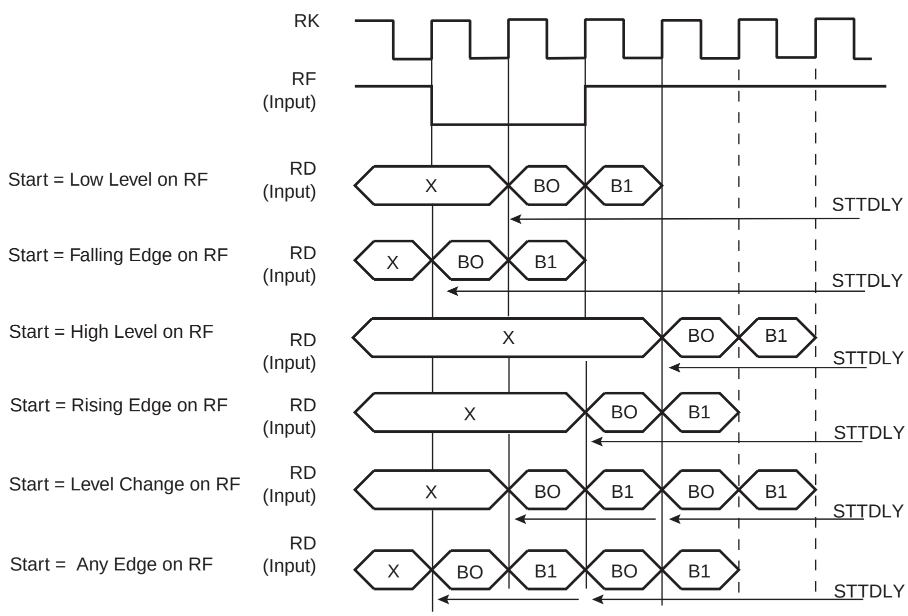


Figure 27-11. Receive Pulse/Edge Start Modes



27.6.5 Frame Sync

The Transmitter and Receiver Frame Sync pins, TF and RF, can be programmed to generate different kinds of frame synchronization signals. The Frame Sync Output Selection (FSOS) field in the Receive Frame Mode Register (SSC_RFMR) and in the Transmit Frame Mode Register (SSC_TFMR) are used to select the required waveform.

- Programmable low or high levels during data transfer are supported.
- Programmable high levels before the start of data transfers or toggling are also supported.

If a pulse waveform is selected, the Frame Sync Length (FSLEN) field in SSC_RFMR and SSC_TFMR programs the length of the pulse, from 1 bit time up to 256 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.

27.6.5.1 Frame Sync Data

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

During the Frame Sync signal, the Receiver can sample the RD line and store the data in the Receive Sync Holding Register and the transmitter can transfer Transmit Sync Holding Register in the Shifter Register. The data length to be sampled/shifted out during the Frame Sync signal is programmed by the FSLEN field in SSC_RFMR/SSC_TFMR and has a maximum value of 256.

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.

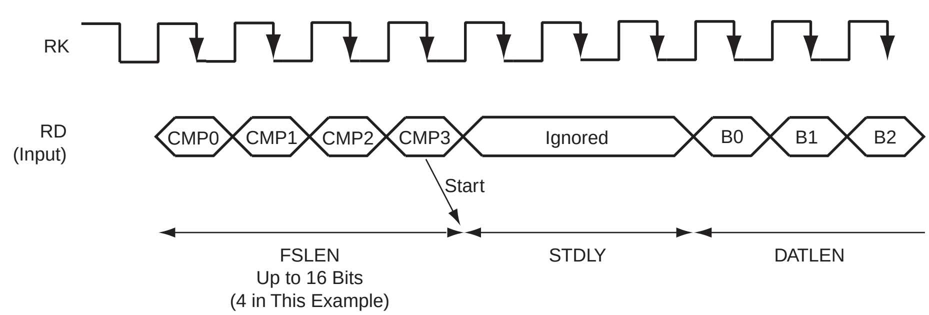
The Transmit Frame Sync Operation is performed by the transmitter only if the bit Frame Sync Data Enable (FSDEN) in SSC_TFMR is set. If the Frame Sync length is equal to or lower than the delay between the start event and the actual data transmission, the normal transmission has priority and the data contained in the Transmit Sync Holding Register is transferred in the Transmit Register, then shifted out.

27.6.5.2 Frame Sync Edge Detection

The Frame Sync Edge detection is programmed by the FSEDGE field in SSC_RFMR/SSC_TFMR. This sets the corresponding flags RXSYN/TXSYN in the SSC Status Register (SSC_SR) on frame synchro edge detection (signals RF/TF).

27.6.6 Receive Compare Modes

Figure 27-12. Receive Compare Modes



27.6.6.1 Compare Functions

Length of the comparison patterns (Compare 0, Compare 1) and thus the number of bits they are compared to is defined by FSLEN, but with a maximum value of 256 bits. Comparison is always done by comparing the last bits received with the comparison pattern. Compare 0 can be one start event of the Receiver. In this case, the receiver compares at each new sample the last bits received at the Compare 0 pattern contained in the Compare 0 Register (SSC_RC0R). When this start event is selected, the user can program the Receiver to start a new data transfer either by writing a new Compare 0, or by receiving continuously until Compare 1 occurs. This selection is done with the bit (STOP) in SSC_RCMR.

27.6.7 Data Format

The data framing format of both the transmitter and the receiver are programmable through the Transmitter Frame Mode Register (SSC_TFMR) and the Receiver Frame Mode Register (SSC_RFMR). In either case, the user can independently select:

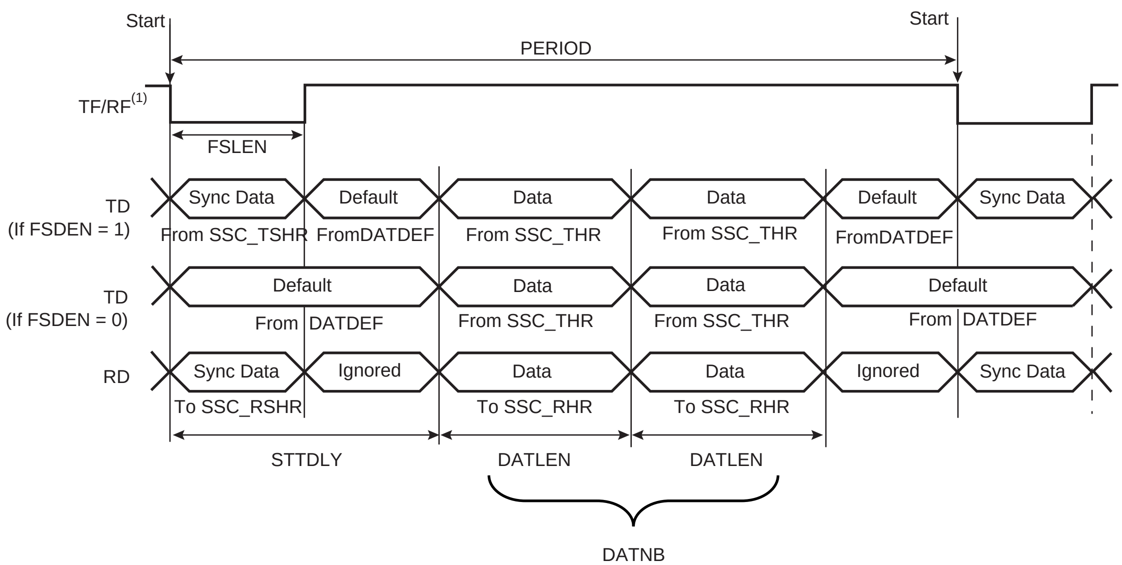
- the event that starts the data transfer (START)
- the delay in number of bit periods between the start event and the first data bit (STTDLY)
- the length of the data (DATLEN)
- the number of data to be transferred for each start event (DATNB).
- the length of synchronization transferred for each start event (FSLEN)
- the bit sense: most or lowest significant bit first (MSBF)

Additionally, the transmitter can be used to transfer synchronization and select the level driven on the TD pin while not in data transfer operation. This is done respectively by the Frame Sync Data Enable (FSDEN) and by the Data Default Value (DATDEF) bits in SSC_TFMR.

Table 27-3. Data Frame Registers

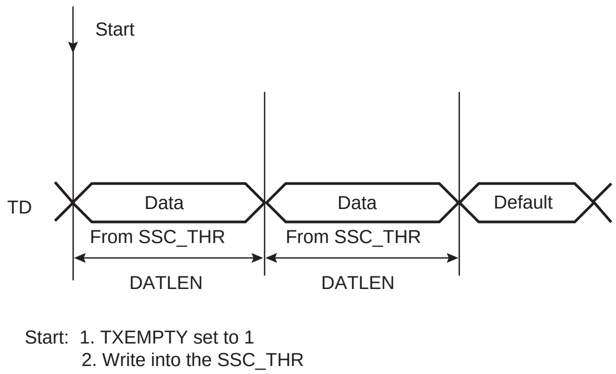
Transmitter	Receiver	Field	Length	Comment
SSC_TFMR	SSC_RFMR	DATLEN	Up to 32	Size of word
SSC_TFMR	SSC_RFMR	DATNB	Up to 16	Number of words transmitted in frame
SSC_TFMR	SSC_RFMR	MSBF		Most significant bit first
SSC_TFMR	SSC_RFMR	FSLEN	Up to 256	Size of Synchro data register
SSC_TFMR		DATDEF	0 or 1	Data 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 27-13. Transmit and Receive Frame Format in Edge/Pulse Start Modes



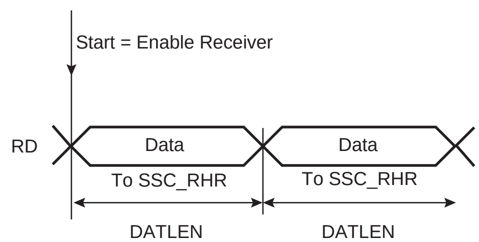
Note: 1. Example of input on falling edge of TF/RF.

Figure 27-14. Transmit Frame Format in Continuous Mode



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

Figure 27-15. Receive Frame Format in Continuous Mode



Note: 1. STTDLY is set to 0.

27.6.8 Loop Mode

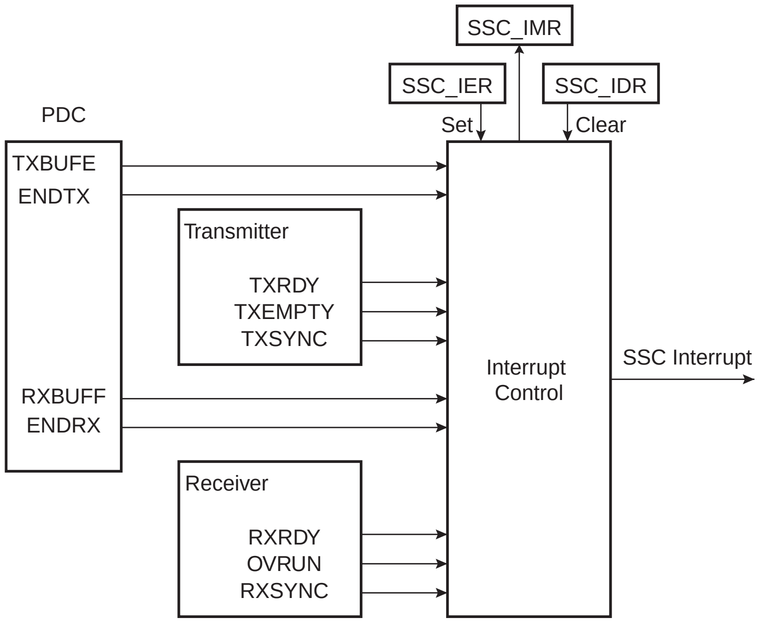
The receiver can be programmed to receive transmissions from the transmitter. This is done by setting the Loop Mode (LOOP) bit in SSC_RFMR. In this case, RD is connected to TD, RF is connected to TF and RK is connected to TK.

27.6.9 Interrupt

Most bits in SSC_SR have a corresponding bit in interrupt management registers.

The SSC can be programmed to generate an interrupt when it detects an event. The interrupt is controlled by writing SSC_IER (Interrupt Enable Register) and SSC_IDR (Interrupt Disable Register) These registers enable and disable, respectively, the corresponding interrupt by setting and clearing the corresponding bit in SSC_IMR (Interrupt Mask Register), which controls the generation of interrupts by asserting the SSC interrupt line connected to the AIC.

Figure 27-16. Interrupt Block Diagram



27.7 SSC Application Examples

The SSC can support several serial communication modes used in audio or high speed serial links. Some standard applications are shown in the following figures. All serial link applications supported by the SSC are not listed here.

Figure 27-17. Audio Application Block Diagram

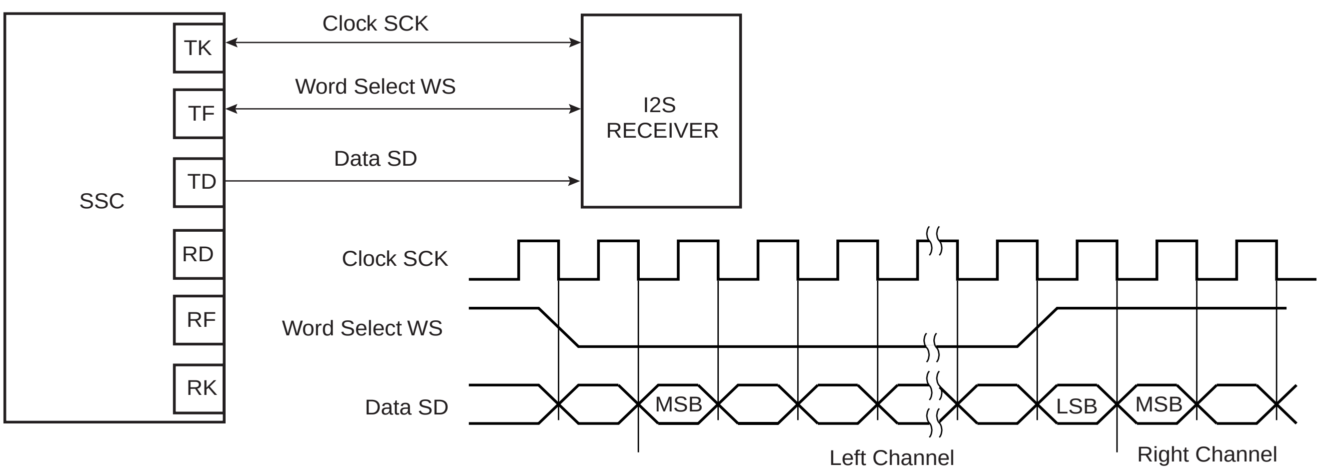


Figure 27-18. Codec Application Block Diagram

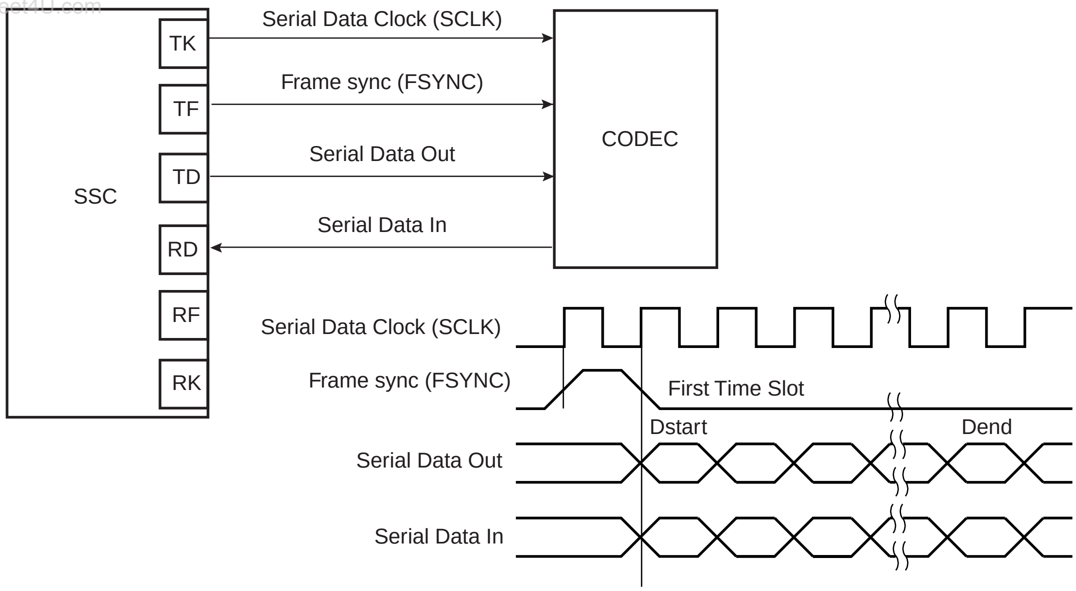
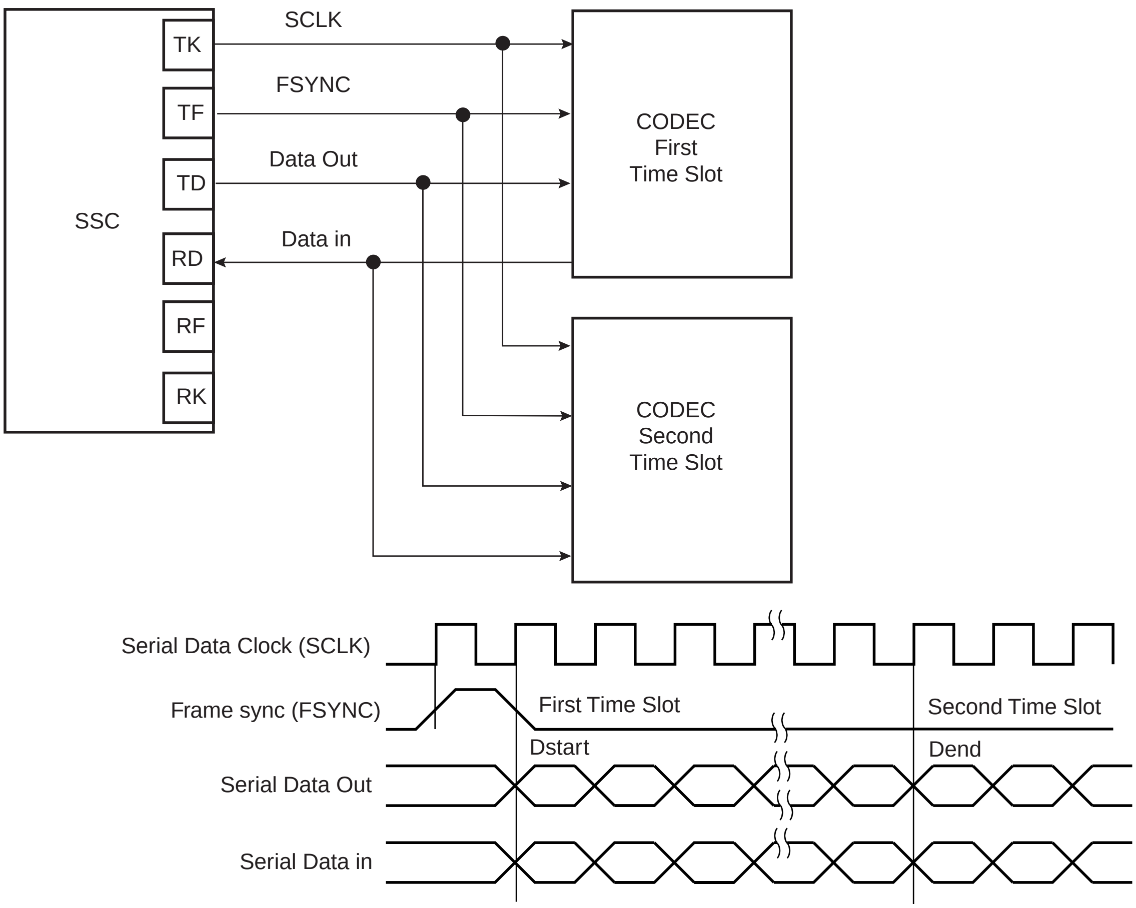


Figure 27-19. Time Slot Application Block Diagram



27.8 Synchronous Serial Controller (SSC) User Interface

Table 27-4. Register Mapping

Offset	Register	Register Name	Access	Reset
0x0	Control Register	SSC_CR	Write	–
0x4	Clock Mode Register	SSC_CMR	Read/Write	0x0
0x8	Reserved	–	–	–
0xC	Reserved	–	–	–
0x10	Receive Clock Mode Register	SSC_RCMR	Read/Write	0x0
0x14	Receive Frame Mode Register	SSC_RFMR	Read/Write	0x0
0x18	Transmit Clock Mode Register	SSC_TCMR	Read/Write	0x0
0x1C	Transmit Frame Mode Register	SSC_TFMR	Read/Write	0x0
0x20	Receive Holding Register	SSC_RHR	Read	0x0
0x24	Transmit Holding Register	SSC_THR	Write	–
0x28	Reserved	–	–	–
0x2C	Reserved	–	–	–
0x30	Receive Sync. Holding Register	SSC_RSHR	Read	0x0
0x34	Transmit Sync. Holding Register	SSC_TSHR	Read/Write	0x0
0x38	Receive Compare 0 Register	SSC_RC0R	Read/Write	0x0
0x3C	Receive Compare 1 Register	SSC_RC1R	Read/Write	0x0
0x40	Status Register	SSC_SR	Read	0x000000CC
0x44	Interrupt Enable Register	SSC_IER	Write	–
0x48	Interrupt Disable Register	SSC_IDR	Write	–
0x4C	Interrupt Mask Register	SSC_IMR	Read	0x0
0x50-0xFC	Reserved	–	–	–
0x100- 0x124	Reserved for Peripheral Data Controller (PDC)	–	–	–

27.8.1 SSC Control Register

Name: SSC_CR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
SWRST	–	–	–	–	–	TXDIS	TXEN

7	6	5	4	3	2	1	0
–	–	–	–	–	–	RXDIS	RXEN

- **RXEN: Receive Enable**

0: No effect.

1: Enables Receive if RXDIS is not set.

- **RXDIS: Receive Disable**

0: No effect.

1: Disables Receive. If a character is currently being received, disables at end of current character reception.

- **TXEN: Transmit Enable**

0: No effect.

1: Enables Transmit if TXDIS is not set.

- **TXDIS: Transmit Disable**

0: No effect.

1: Disables Transmit. If a character is currently being transmitted, disables at end of current character transmission.

- **SWRST: Software Reset**

0: No effect.

1: Performs a software reset. Has priority on any other bit in SSC_CR.

27.8.2 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.

27.8.3 SSC Receive Clock Mode Register

Name: SSC_RCMR

Access Type: Read/Write

31	30	29	28	27	26	25	24
PERIOD							

23	22	21	20	19	18	17	16
STDDLY							

15	14	13	12	11	10	9	8
–	–	–	STOP	START			

7	6	5	4	3	2	1	0
CKG		CKI	CKO			CKS	

- **CKS: Receive Clock Selection**

CKS	Selected Receive Clock
0x0	Divided Clock
0x1	TK Clock signal
0x2	RK pin
0x3	Reserved

- **CKO: Receive Clock Output Mode Selection**

CKO	Receive Clock Output Mode	RK pin
0x0	None	Input-only
0x1	Continuous Receive Clock	Output
0x2	Receive Clock only during data transfers	Output
0x3-0x7	Reserved	

- **CKI: Receive Clock Inversion**

0: The data inputs (Data and Frame Sync signals) are sampled on Receive Clock falling edge. The Frame Sync signal output is shifted out on Receive Clock rising edge.

1: The data inputs (Data and Frame Sync signals) are sampled on Receive Clock rising edge. The Frame Sync signal output is shifted out on Receive Clock falling edge.

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

- **CKG: Receive Clock Gating Selection**

CKG	Receive Clock Gating
0x0	None, continuous clock
0x1	Receive Clock enabled only if RF Low
0x2	Receive Clock enabled only if RF High
0x3	Reserved

- **START: Receive Start Selection**

START	Receive Start
0x0	Continuous, as soon as the receiver is enabled, and immediately after the end of transfer of the previous data.
0x1	Transmit start
0x2	Detection of a low level on RF signal
0x3	Detection of a high level on RF signal
0x4	Detection of a falling edge on RF signal
0x5	Detection of a rising edge on RF signal
0x6	Detection of any level change on RF signal
0x7	Detection of any edge on RF signal
0x8	Compare 0
0x9-0xF	Reserved

- **STOP: Receive Stop Selection**

0: After completion of a data transfer when starting with a Compare 0, the receiver stops the data transfer and waits for a new compare 0.

1: After starting a receive with a Compare 0, the receiver operates in a continuous mode until a Compare 1 is detected.

- **STTDLY: Receive Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of reception. When the Receiver is programmed to start synchronously with the Transmitter, the delay is also applied.

Note: It is very important that STTDLY be set carefully. If STTDLY must be set, it should be done in relation to TAG (Receive Sync Data) reception.

- **PERIOD: Receive Period Divider Selection**

This field selects the divider to apply to the selected Receive Clock in order to generate a new Frame Sync Signal. If 0, no PERIOD signal is generated. If not 0, a PERIOD signal is generated each 2 x (PERIOD+1) Receive Clock.

27.8.4 SSC Receive Frame Mode Register

Name: SSC_RFMR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
–	FSOS			FSLEN			

15	14	13	12	11	10	9	8
FSLEN				DATNB			

7	6	5	4	3	2	1	0
MSBF	–	LOOP	DATLEN				

- **DATLEN: Data Length**

0: Forbidden value (1-bit data length not supported).
Any other value: The bit stream contains DATLEN + 1 data bits. Moreover, it defines the transfer size performed by the PDC2 assigned to the Receiver. If DATLEN is lower or equal to 7, data transfers are in bytes. If DATLEN is between 8 and 15 (included), half-words are transferred, and for any other value, 32-bit words are transferred.

- **LOOP: Loop Mode**

0: Normal operating mode.

1: RD is driven by TD, RF is driven by TF and TK drives RK.

- **MSBF: Most Significant Bit First**

0: The lowest significant bit of the data register is sampled first in the bit stream.

1: The most significant bit of the data register is sampled first in the bit stream.

- **DATNB: Data Number per Frame**

This field defines the number of data words to be received after each transfer start, which is equal to (DATNB + 1).

- **FSLEN: Receive Frame Sync Length**

This field defines the number of bits sampled and stored in the Receive Sync Data Register. When this mode is selected by the START field in the Receive Clock Mode Register, it also determines the length of the sampled data to be compared to the Compare 0 or Compare 1 register.

This field is used to determine the pulse length of the Receive Frame Sync signal.

Pulse length is equal to FSLEN + 1 Receive Clock periods.

- **FSOS: Receive Frame Sync Output Selection**

FSOS	Selected Receive Frame Sync Signal	RF Pin
0x0	None	Input-only
0x1	Negative Pulse	Output
0x2	Positive Pulse	Output
0x3	Driven Low during data transfer	Output
0x4	Driven High during data transfer	Output
0x5	Toggling at each start of data transfer	Output
0x6-0x7	Reserved	Undefined

- **FSEDGE: Frame Sync Edge Detection**

Determines which edge on Frame Sync will generate the interrupt RXSYN in the SSC Status Register.

FSEDGE	Frame Sync Edge Detection
0x0	Positive Edge Detection
0x1	Negative Edge Detection

27.8.5 SSC Transmit Clock Mode Register

Name: SSC_TCMR

Access Type: Read/Write

31	30	29	28	27	26	25	24
PERIOD							

23	22	21	20	19	18	17	16
STTDLY							

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

7	6	5	4	3	2	1	0
CKG		CKI	CKO			CKS	

- **CKS: Transmit Clock Selection**

CKS	Selected Transmit Clock
0x0	Divided Clock
0x1	RK Clock signal
0x2	TK Pin
0x3	Reserved

- **CKO: Transmit Clock Output Mode Selection**

CKO	Transmit Clock Output Mode	TK pin
0x0	None	Input-only
0x1	Continuous Transmit Clock	Output
0x2	Transmit Clock only during data transfers	Output
0x3-0x7	Reserved	

- **CKI: Transmit Clock Inversion**

0: The data outputs (Data and Frame Sync signals) are shifted out on Transmit Clock falling edge. The Frame sync signal input is sampled on Transmit clock rising edge.

1: The data outputs (Data and Frame Sync signals) are shifted out on Transmit Clock rising edge. The Frame sync signal input is sampled on Transmit clock falling edge.

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

- **CKG: Transmit Clock Gating Selection**

CKG	Transmit Clock Gating
0x0	None, continuous clock
0x1	Transmit Clock enabled only if TF Low
0x2	Transmit Clock enabled only if TF High
0x3	Reserved

- **START: Transmit Start Selection**

START	Transmit Start
0x0	Continuous, as soon as a word is written in the SSC_THR Register (if Transmit is enabled), and immediately after the end of transfer of the previous data.
0x1	Receive start
0x2	Detection of a low level on TF signal
0x3	Detection of a high level on TF signal
0x4	Detection of a falling edge on TF signal
0x5	Detection of a rising edge on TF signal
0x6	Detection of any level change on TF signal
0x7	Detection of any edge on TF signal
0x8 - 0xF	Reserved

- **STTDLY: Transmit Start Delay**

If STTDLY is not 0, a delay of STTDLY clock cycles is inserted between the start event and the actual start of transmission of data. When the Transmitter is programmed to start synchronously with the Receiver, the delay is also applied.

Note: STTDLY must be set carefully. If STTDLY is too short in respect to TAG (Transmit Sync Data) emission, data is emitted instead of the end of TAG.

- **PERIOD: Transmit Period Divider Selection**

This field selects the divider to apply to the selected Transmit Clock to generate a new Frame Sync Signal. If 0, no period signal is generated. If not 0, a period signal is generated at each 2 x (PERIOD+1) Transmit Clock.

27.8.6 SSC Transmit Frame Mode Register

Name: SSC_TFMR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
FSDEN	FSOS			FSLEN			

15	14	13	12	11	10	9	8
FSLEN				DATNB			

7	6	5	4	3	2	1	0
MSBF	–	DATDEF	DATLEN				

- **DATLEN: Data Length**

0: Forbidden value (1-bit data length not supported).

Any other value: The bit stream contains DATLEN + 1 data bits. Moreover, it defines the transfer size performed by the PDC2 assigned to the Transmit. If DATLEN is lower or equal to 7, data transfers are bytes, if DATLEN is between 8 and 15 (included), half-words are transferred, and for any other value, 32-bit words are transferred.

- **DATDEF: Data Default Value**

This bit defines the level driven on the TD pin while out of transmission. Note that if the pin is defined as multi-drive by the PIO Controller, the pin is enabled only if the SCC TD output is 1.

- **MSBF: Most Significant Bit First**

0: The lowest significant bit of the data register is shifted out first in the bit stream.

1: The most significant bit of the data register is shifted out first in the bit stream.

- **DATNB: Data Number per frame**

This field defines the number of data words to be transferred after each transfer start, which is equal to (DATNB +1).

- **FSLEN: Transmit Frame Sync Length**

This field defines the length of the Transmit Frame Sync signal and the number of bits shifted out from the Transmit Sync Data Register if FSDEN is 1.

This field is used to determine the pulse length of the Transmit Frame Sync signal.

Pulse length is equal to FSLEN + 1 Transmit Clock periods.

- **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 will generate the interrupt TXSYN (Status Register).

FSEDGE	Frame Sync Edge Detection
0x0	Positive Edge Detection
0x1	Negative Edge Detection

27.8.7 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.

27.8.8 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.

27.8.9 SSC Receive Synchronization Holding Register

Name: SSC_RSHR

Access Type: Read-only

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

- **RSDAT: Receive Synchronization Data**

27.8.10 SSC Transmit Synchronization Holding Register

Name: SSC_TSHR

Access Type: Read/Write

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

- **TSDAT: Transmit Synchronization Data**

27.8.11 SSC Receive Compare 0 Register

Name: SSC_RC0R

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
CP0							

7	6	5	4	3	2	1	0
CP0							

- **CP0: Receive Compare Data 0**

27.8.12 SSC Receive Compare 1 Register

Name: SSC_RC1R

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
CP1							

7	6	5	4	3	2	1	0
CP1							

- **CP1: Receive Compare Data 1**

27.8.13 SSC Status Register

Name: SSC_SR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0

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

- **TXRDY: Transmit Ready**

0: Data has been loaded in SSC_THR and is waiting to be loaded in the Transmit Shift Register (TSR).

1: SSC_THR is empty.

- **TXEMPTY: Transmit Empty**

0: Data remains in SSC_THR or is currently transmitted from TSR.

1: Last data written in SSC_THR has been loaded in TSR and last data loaded in TSR has been transmitted.

- **ENDTX: End of Transmission**

0: The register SSC_TCR has not reached 0 since the last write in SSC_TCR or SSC_TNCR.

1: The register SSC_TCR has reached 0 since the last write in SSC_TCR or SSC_TNCR.

- **TXBUFE: Transmit Buffer Empty**

0: SSC_TCR or SSC_TNCR have a value other than 0.

1: Both SSC_TCR and SSC_TNCR have a value of 0.

- **RXRDY: Receive Ready**

0: SSC_RHR is empty.

1: Data has been received and loaded in SSC_RHR.

- **OVRUN: Receive Overrun**

0: No data has been loaded in SSC_RHR while previous data has not been read since the last read of the Status Register.

1: Data has been loaded in SSC_RHR while previous data has not yet been read since the last read of the Status Register.

- **ENDRX: End of Reception**

0: Data is written on the Receive Counter Register or Receive Next Counter Register.

1: End of PDC transfer when Receive Counter Register has arrived at zero.

- **RXBUFF: Receive Buffer Full**

0: SSC_RCR or SSC_RNCR have a value other than 0.

1: Both SSC_RCR and SSC_RNCR have a value of 0.

- **CP0: Compare 0**

0: A compare 0 has not occurred since the last read of the Status Register.

1: A compare 0 has occurred since the last read of the Status Register.

- **CP1: Compare 1**

0: A compare 1 has not occurred since the last read of the Status Register.

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

- **TXSYN: Transmit Sync**

0: A Tx Sync has not occurred since the last read of the Status Register.

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

- **RXSYN: Receive Sync**

0: An Rx Sync has not occurred since the last read of the Status Register.

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

- **TXEN: Transmit Enable**

0: Transmit is disabled.

1: Transmit is enabled.

- **RXEN: Receive Enable**

0: Receive is disabled.

1: Receive is enabled.

27.8.14 SSC Interrupt Enable Register

Name: SSC_IER

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0

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

- **TXRDY: Transmit Ready Interrupt Enable**

0: No effect.

1: Enables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Enable**

0: No effect.

1: Enables the Transmit Empty Interrupt.

- **ENDTX: End of Transmission Interrupt Enable**

0: No effect.

1: Enables the End of Transmission Interrupt.

- **TXBUFE: Transmit Buffer Empty Interrupt Enable**

0: No effect.

1: Enables the Transmit Buffer Empty Interrupt

- **RXRDY: Receive Ready Interrupt Enable**

0: No effect.

1: Enables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Enable**

0: No effect.

1: Enables the Receive Overrun Interrupt.

- **ENDRX: End of Reception Interrupt Enable**

0: No effect.

1: Enables the End of Reception Interrupt.

- **RXBUFF: Receive Buffer Full Interrupt Enable**

0: No effect.

1: Enables the Receive Buffer Full Interrupt.

- **CP0: Compare 0 Interrupt Enable**

0: No effect.

1: Enables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Enable**

0: No effect.

1: Enables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0: No effect.

1: Enables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0: No effect.

1: Enables the Rx Sync Interrupt.

27.8.15 SSC Interrupt Disable Register

Name: SSC_IDR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0

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

- **TXRDY: Transmit Ready Interrupt Disable**

0: No effect.

1: Disables the Transmit Ready Interrupt.

- **TXEMPTY: Transmit Empty Interrupt Disable**

0: No effect.

1: Disables the Transmit Empty Interrupt.

- **ENDTX: End of Transmission Interrupt Disable**

0: No effect.

1: Disables the End of Transmission Interrupt.

- **TXBUFE: Transmit Buffer Empty Interrupt Disable**

0: No effect.

1: Disables the Transmit Buffer Empty Interrupt.

- **RXRDY: Receive Ready Interrupt Disable**

0: No effect.

1: Disables the Receive Ready Interrupt.

- **OVRUN: Receive Overrun Interrupt Disable**

0: No effect.

1: Disables the Receive Overrun Interrupt.

- **ENDRX: End of Reception Interrupt Disable**

0: No effect.

1: Disables the End of Reception Interrupt.

- **RXBUFF: Receive Buffer Full Interrupt Disable**

0: No effect.

1: Disables the Receive Buffer Full Interrupt.

- **CP0: Compare 0 Interrupt Disable**

0: No effect.

1: Disables the Compare 0 Interrupt.

- **CP1: Compare 1 Interrupt Disable**

0: No effect.

1: Disables the Compare 1 Interrupt.

- **TXSYN: Tx Sync Interrupt Enable**

0: No effect.

1: Disables the Tx Sync Interrupt.

- **RXSYN: Rx Sync Interrupt Enable**

0: No effect.

1: Disables the Rx Sync Interrupt.

27.8.16 SSC Interrupt Mask Register

Name: SSC_IMR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	–	–	RXSYN	TXSYN	CP1	CP0

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

- **TXRDY: Transmit Ready Interrupt Mask**

0: The Transmit Ready Interrupt is disabled.

1: The Transmit Ready Interrupt is enabled.

- **TXEMPTY: Transmit Empty Interrupt Mask**

0: The Transmit Empty Interrupt is disabled.

1: The Transmit Empty Interrupt is enabled.

- **ENDTX: End of Transmission Interrupt Mask**

0: The End of Transmission Interrupt is disabled.

1: The End of Transmission Interrupt is enabled.

- **TXBUFE: Transmit Buffer Empty Interrupt Mask**

0: The Transmit Buffer Empty Interrupt is disabled.

1: The Transmit Buffer Empty Interrupt is enabled.

- **RXRDY: Receive Ready Interrupt Mask**

0: The Receive Ready Interrupt is disabled.

1: The Receive Ready Interrupt is enabled.

- **OVRUN: Receive Overrun Interrupt Mask**

0: The Receive Overrun Interrupt is disabled.

1: The Receive Overrun Interrupt is enabled.

- **ENDRX: End of Reception Interrupt Mask**

0: The End of Reception Interrupt is disabled.

1: The End of Reception Interrupt is enabled.

- **RXBUFF: Receive Buffer Full Interrupt Mask**

0: The Receive Buffer Full Interrupt is disabled.

1: The Receive Buffer Full Interrupt is enabled.

- **CP0: Compare 0 Interrupt Mask**

0: The Compare 0 Interrupt is disabled.

1: The Compare 0 Interrupt is enabled.

- **CP1: Compare 1 Interrupt Mask**

0: The Compare 1 Interrupt is disabled.

1: The Compare 1 Interrupt is enabled.

- **TXSYN: Tx Sync Interrupt Mask**

0: The Tx Sync Interrupt is disabled.

1: The Tx Sync Interrupt is enabled.

- **RXSYN: Rx Sync Interrupt Mask**

0: The Rx Sync Interrupt is disabled.

1: The Rx Sync Interrupt is enabled.

28. Timer Counter (TC)

28.1 Description

The Timer Counter (TC) includes three identical 16-bit Timer Counter channels.

Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation.

Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user. Each channel drives an internal interrupt signal which can be programmed to generate processor interrupts.

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

The Block Control Register allows the three channels to be started simultaneously with the same instruction.

The Block Mode Register defines the external clock inputs for each channel, allowing them to be chained.

Table 28-1 gives the assignment of the device Timer Counter clock inputs common to Timer Counter 0 to 2

Table 28-1. Timer Counter Clock Assignment

Name	Definition
TIMER_CLOCK1	MCK/2
TIMER_CLOCK2	MCK/8
TIMER_CLOCK3	MCK/32
TIMER_CLOCK4	MCK/128
TIMER_CLOCK5	SLCK

28.2 Block Diagram

Figure 28-1. Timer Counter Block Diagram

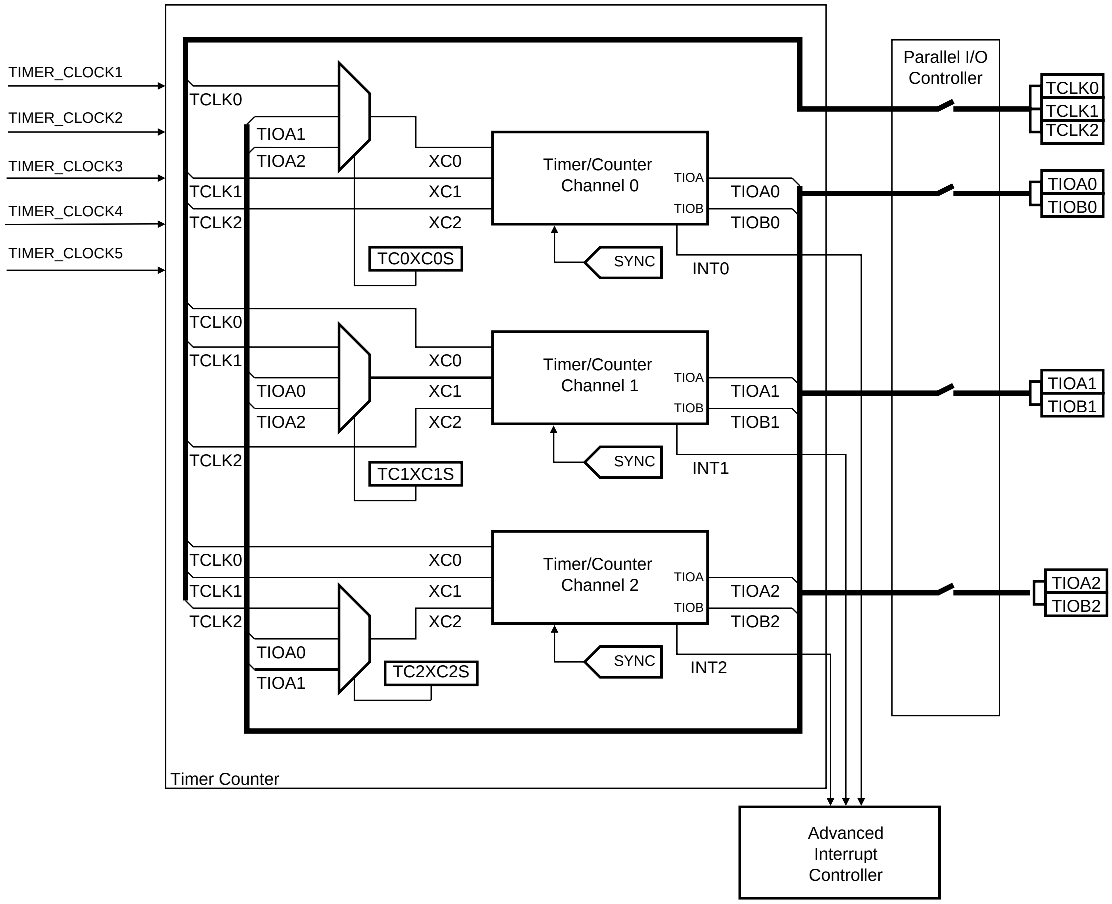


Table 28-2. Signal Name Description

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Output
	TIOB	Capture Mode: Timer Counter Input Waveform Mode: Timer Counter Input/Output
	INT	Interrupt Signal Output
	SYNC	Synchronization Input Signal

28.3 Pin Name List

Table 28-3. TC pin list

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

28.4 Product Dependencies

28.4.1 I/O Lines

The pins used for interfacing the compliant external devices may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the TC pins to their peripheral functions.

28.4.2 Power Management

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

28.4.3 Interrupt

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

28.5 Functional Description

28.5.1 TC Description

The three channels of the Timer Counter are independent and identical in operation. The registers for channel programming are listed in Table 28-5 on page 517.

28.5.2 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.

28.5.3 Clock Selection

At block level, input clock signals of each channel can either be connected to the external inputs TCLK0, TCLK1 or TCLK2, or be connected to the internal I/O signals TIOA0, TIOA1 or TIOA2 for chaining by programming the TC_BMR (Block Mode). See Figure 28-2 on page 505.

Each channel can independently select an internal or external clock source for its counter:

- Internal clock signals: TIMER_CLOCK1, TIMER_CLOCK2, TIMER_CLOCK3, TIMER_CLOCK4, TIMER_CLOCK5
- External clock signals: XC0, XC1 or XC2

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

The selected clock can be inverted with the CLKI bit in TC_CMR. This allows counting on the opposite edges of the clock.

The burst function allows the clock to be validated when an external signal is high. The BURST parameter in the Mode Register defines this signal (none, XC0, XC1, XC2). See Figure 28-3 on page 505

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 28-2. Clock Chaining Selection

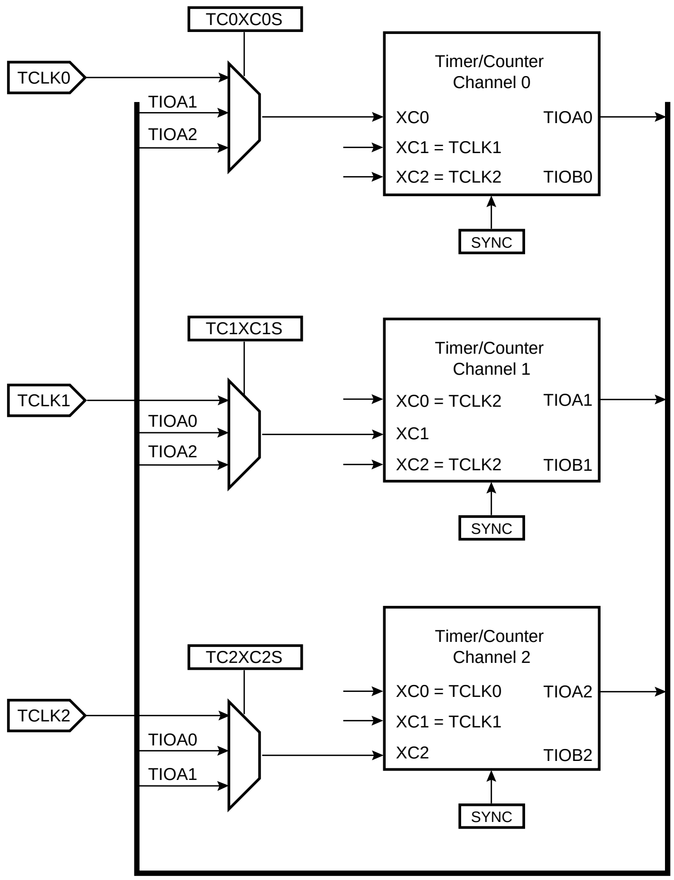
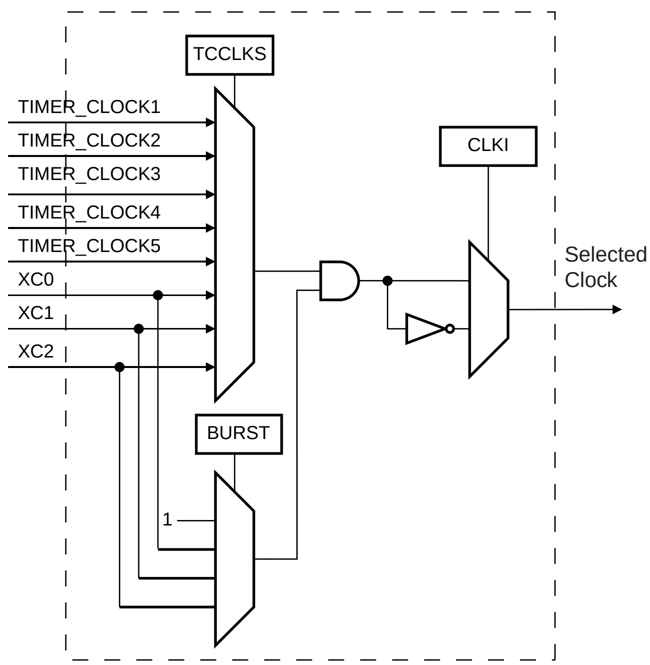


Figure 28-3. Clock Selection

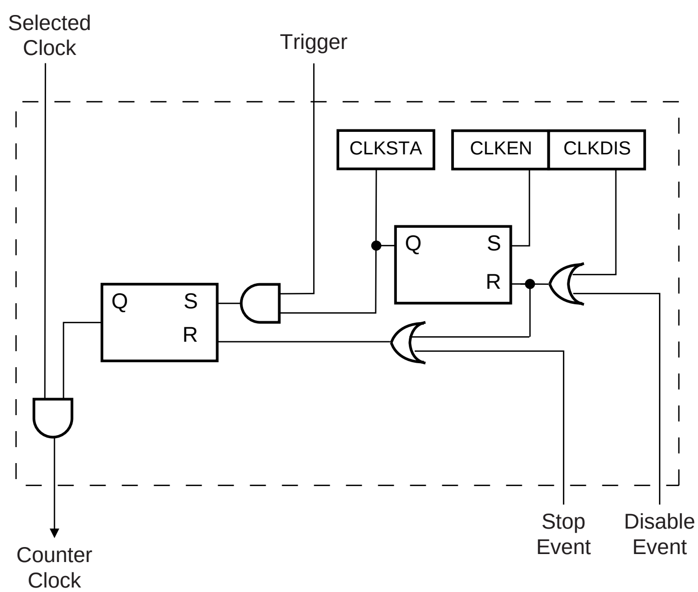


28.5.4 Clock Control

The clock of each counter can be controlled in two different ways: it can be enabled/disabled and started/stopped. See Figure 28-4.

- The clock can be enabled or disabled by the user with the CLKEN and the CLKDIS commands in the Control Register. In Capture Mode it can be disabled by an RB load event if LDBDIS is set to 1 in TC_CMR. In Waveform Mode, it can be disabled by an RC Compare event if CPCDIS is set to 1 in TC_CMR. When disabled, the start or the stop actions have no effect: only a CLKEN command in the Control Register can re-enable the clock. When the clock is enabled, the CLKSTA bit is set in the Status Register.
- The clock can also be started or stopped: a trigger (software, synchro, external or compare) always starts the clock. The clock can be stopped by an RB load event in Capture Mode (LDBSTOP = 1 in TC_CMR) or a RC compare event in Waveform Mode (CPCSTOP = 1 in TC_CMR). The start and the stop commands have effect only if the clock is enabled.

Figure 28-4. Clock Control



28.5.5 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.

28.5.6 Trigger

A trigger resets the counter and starts the counter clock. Three types of triggers are common to both modes, and a fourth external trigger is available to each mode.

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.

28.5.7 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 28-5 shows the configuration of the TC channel when programmed in Capture Mode.

28.5.8 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.

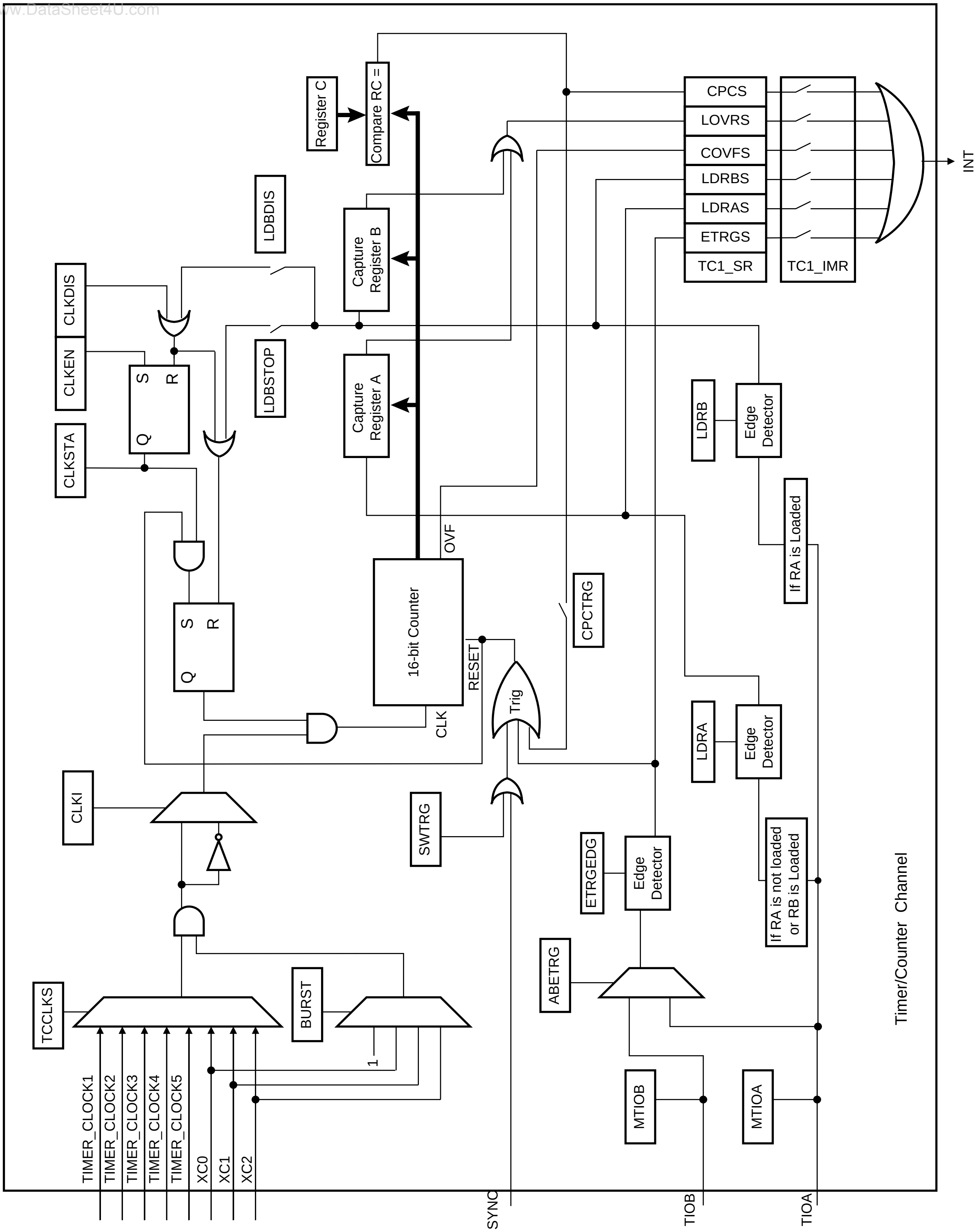
Loading RA or RB before the read of the last value loaded sets the Overrun Error Flag (LOVRS) in TC_SR (Status Register). In this case, the old value is overwritten.

28.5.9 Trigger Conditions

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

The ABETRG bit in 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 28-5. Capture Mode



28.5.10 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 28-6 shows the configuration of the TC channel when programmed in Waveform Operating Mode.

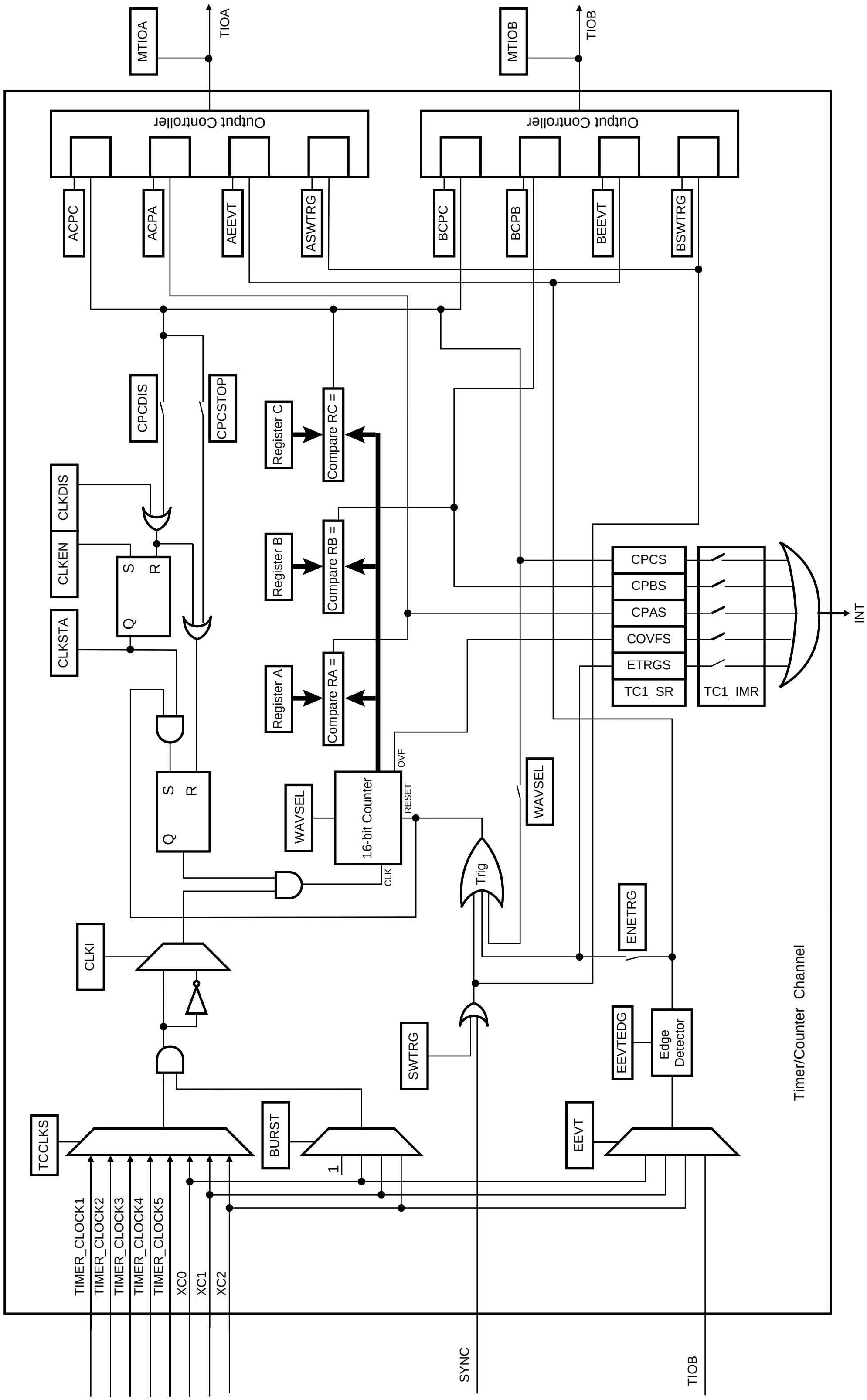
28.5.11 Waveform Selection

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 28-6. Waveform Mode



28.5.11.1 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 28-7.

An external event trigger or a software trigger can reset the value of TC_CV. It is important to note that the trigger may occur at any time. See Figure 28-8.

RC Compare cannot be programmed to generate a trigger in this configuration. At the same time, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 28-7. WAVSEL= 00 without trigger

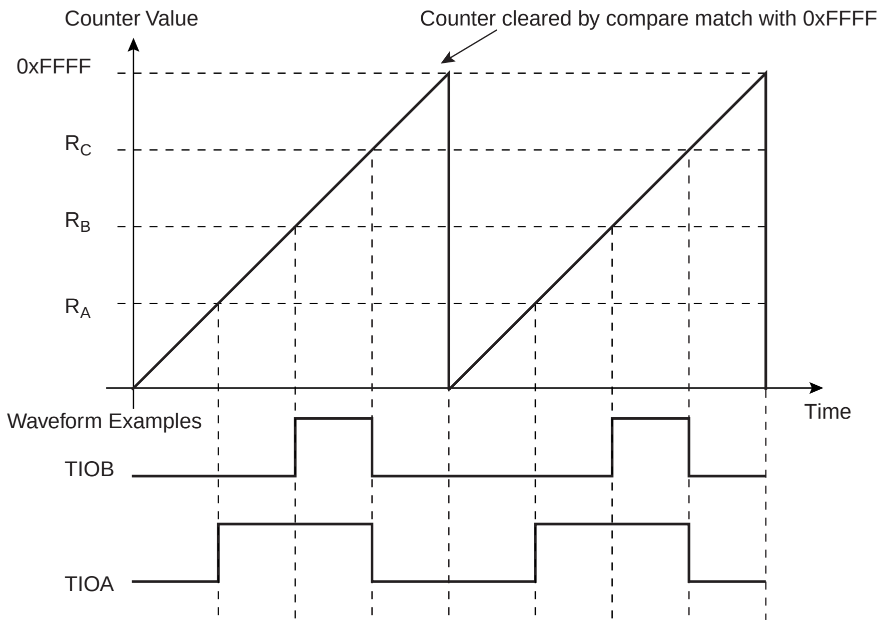
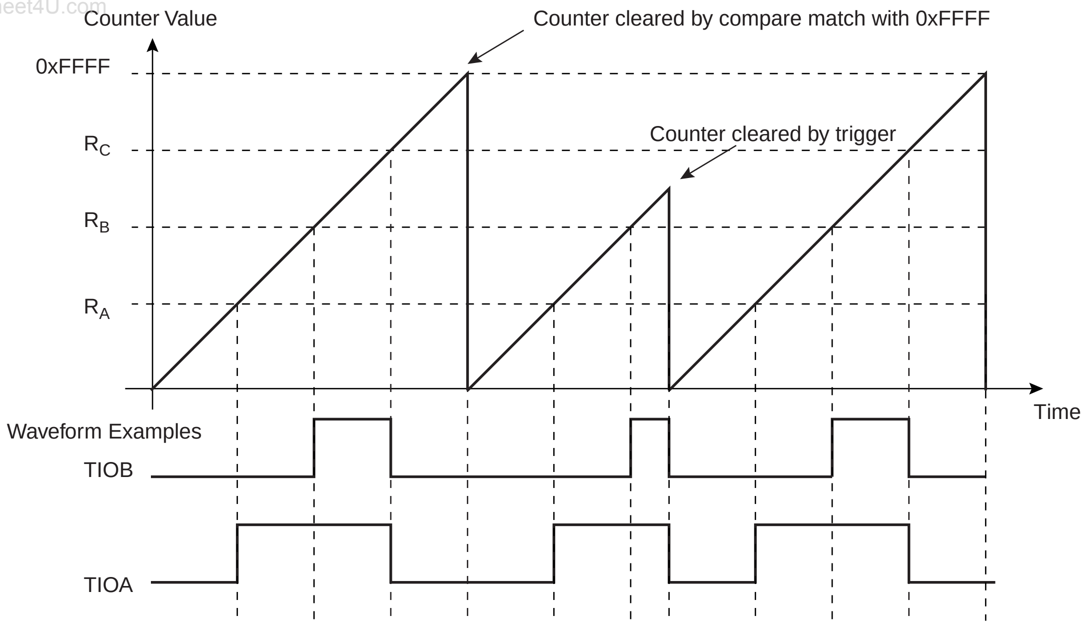


Figure 28-8. WAVSEL= 00 with trigger



28.5.11.2 WAVSEL = 10

When WAVSEL = 10, the value of TC_CV is incremented from 0 to the value of RC, then automatically reset on a RC Compare. Once the value of TC_CV has been reset, it is then incremented and so on. See Figure 28-9.

It is important to note that TC_CV can be reset at any time by an external event or a software trigger if both are programmed correctly. See Figure 28-10.

In addition, RC Compare can stop the counter clock (CPCSTOP = 1 in TC_CMR) and/or disable the counter clock (CPCDIS = 1 in TC_CMR).

Figure 28-9. WAVSEL = 10 Without Trigger

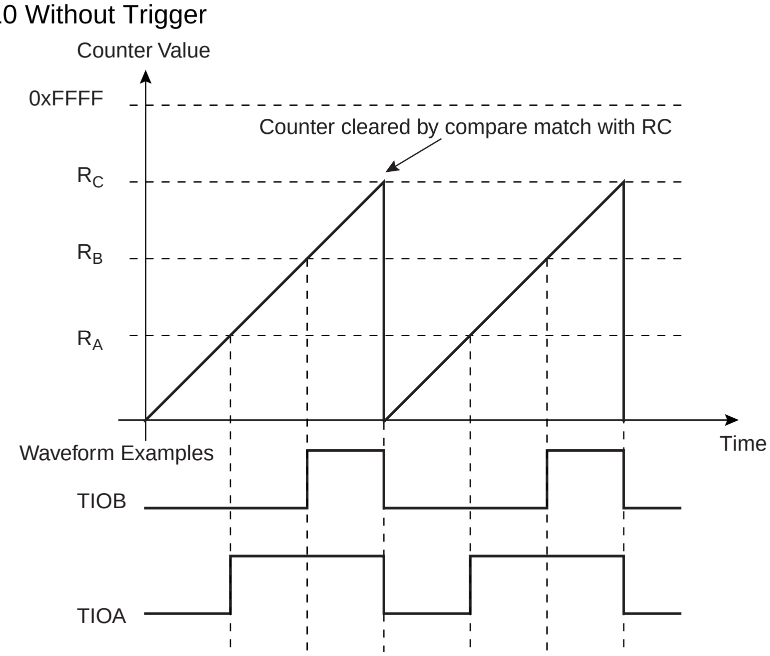
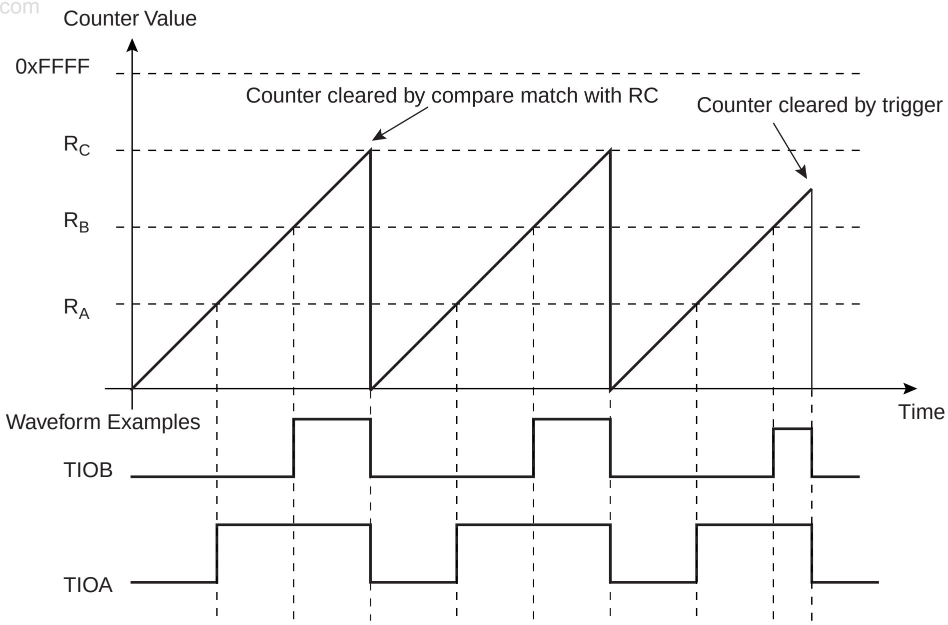


Figure 28-10. WAVSEL = 10 With Trigger



28.5.11.3 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 28-11.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 28-12.

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

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

Figure 28-11. WAVSEL = 01 Without Trigger

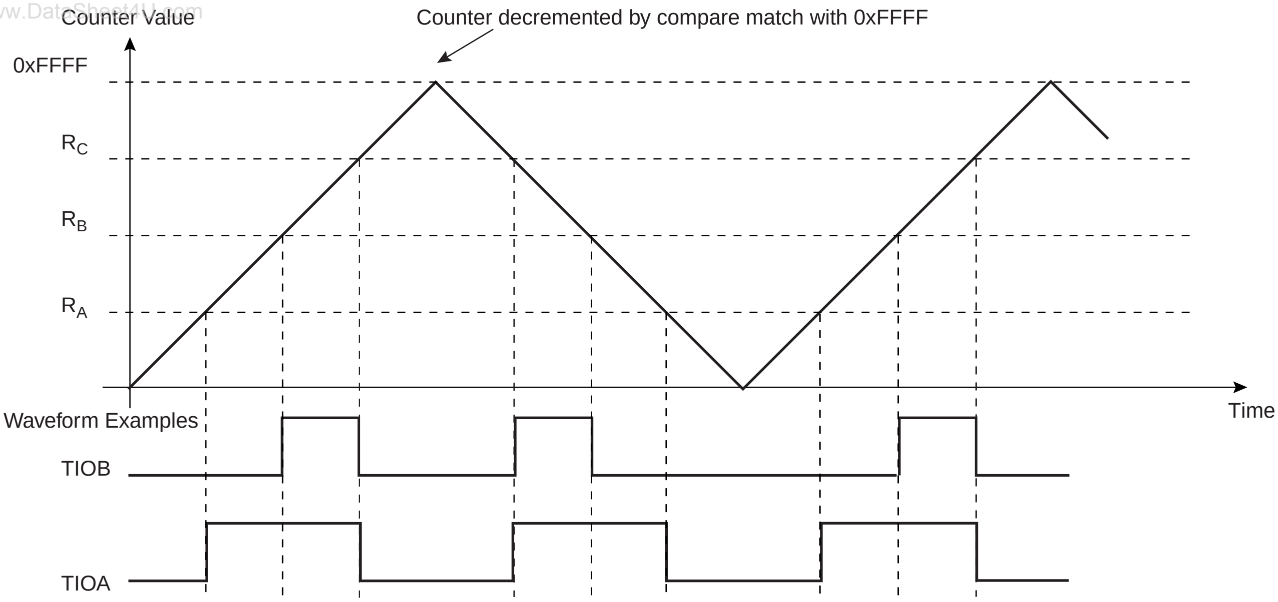
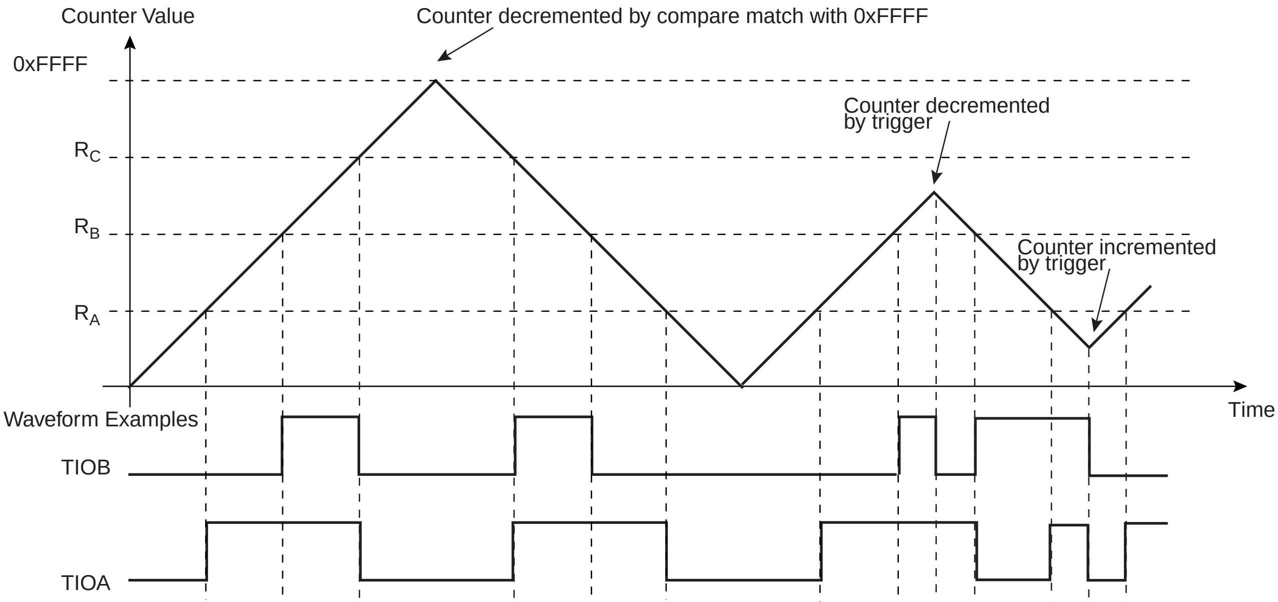


Figure 28-12. WAVSEL = 01 With Trigger



28.5.11.4 WAVSEL = 11

When WAVSEL = 11, the value of TC_CV is incremented from 0 to RC. Once RC is reached, the value of TC_CV is decremented to 0, then re-incremented to RC and so on. See Figure 28-13.

A trigger such as an external event or a software trigger can modify TC_CV at any time. If a trigger occurs while TC_CV is incrementing, TC_CV then decrements. If a trigger is received while TC_CV is decrementing, TC_CV then increments. See Figure 28-14.

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

Figure 28-13. WAVSEL = 11 Without Trigger

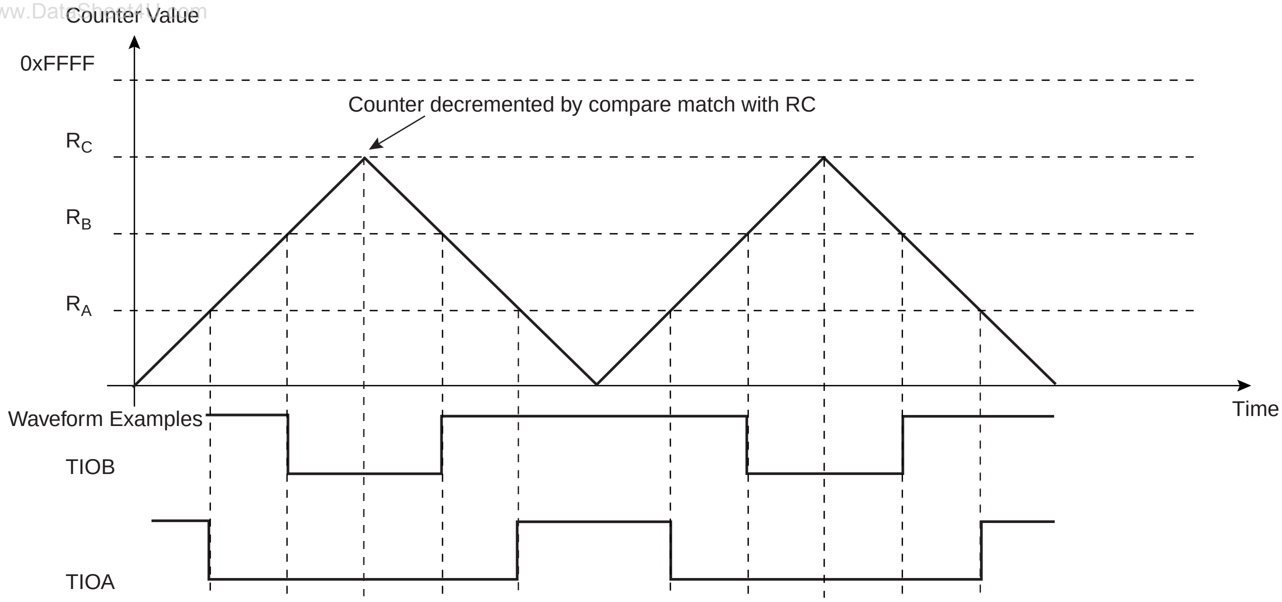
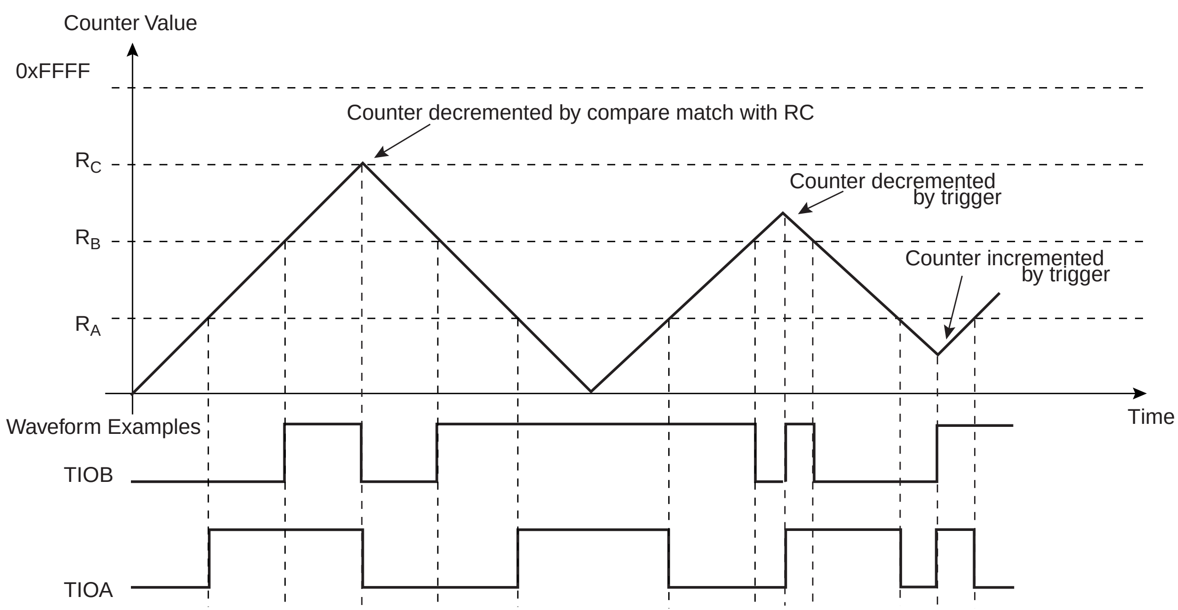


Figure 28-14. WAVSEL = 11 With Trigger



28.5.12 External Event/Trigger Conditions

An external event can be programmed to be detected on one of the clock sources (XC0, XC1, XC2) or TIOB. The external event selected can then be used as a trigger.

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

If TIOB is defined as an external event signal (EEVT = 0), TIOB is no longer used as an output and the compare register B is not used to generate waveforms and subsequently no IRQs. In this case the TC channel can only generate a waveform on TIOA.

When an external event is defined, it can be used as a trigger by setting bit ENETRG in 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.

28.5.13 Output Controller

The output controller defines the output level changes on TIOA and TIOB following an event. TIOB control is used only if TIOB is defined as output (not as an external event).

The following events control TIOA and TIOB: software trigger, external event and RC compare. RA compare controls TIOA and RB compare controls TIOB. Each of these events can be programmed to set, clear or toggle the output as defined in the corresponding parameter in TC_CMR.

28.6 Timer Counter (TC) User Interface

Table 28-4. TC Global Memory Map

Offset	Channel/Register	Name	Access	Reset Value
0x00	TC Channel 0	See Table 28-5		
0x40	TC Channel 1	See Table 28-5		
0x80	TC Channel 2	See Table 28-5		
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 28-5. The offset of each of the channel registers in Table 28-5 is in relation to the offset of the corresponding channel as mentioned in Table 28-5.

Table 28-5. TC Channel Memory Map

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

Notes: 1. Read-only if WAVE = 0

28.6.1 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.

28.6.2 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

28.6.3 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.

28.6.4 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

28.6.5 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 and subsequently no IRQs*.

- **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

28.6.6 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.

28.6.7 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.

28.6.8 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.

28.6.9 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.

28.6.10 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.

28.6.11 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.

28.6.12 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.

28.6.13 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.

29. MultiMedia Card Interface (MCI)

29.1 Description

The MultiMedia Card Interface (MCI) supports the MultiMedia Card (MMC) Specification V3.11, the SDIO Specification V1.1 and the SD Memory Card Specification V1.0.

The MCI includes a command register, response registers, data registers, timeout counters and error detection logic that automatically handle the transmission of commands and, when required, the reception of the associated responses and data with a limited processor overhead.

The MCI supports stream, block and multi-block data read and write, and is compatible with the Peripheral DMA Controller (PDC) channels, minimizing processor intervention for large buffer transfers.

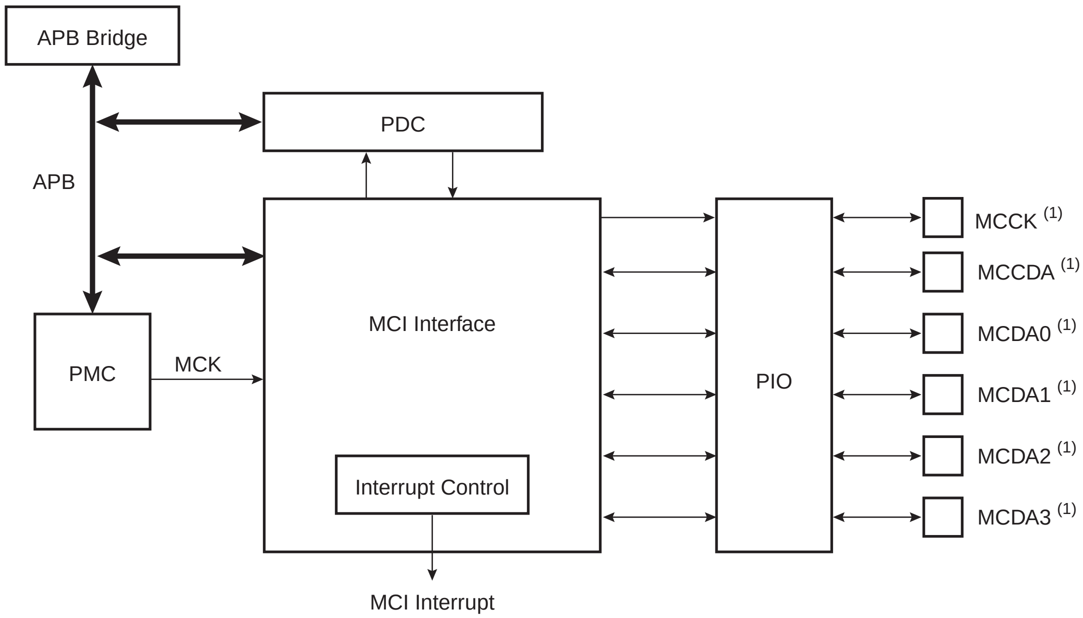
The MCI operates at a rate of up to Master Clock divided by 2 and supports the interfacing of 1 slot(s). Each slot may be used to interface with a MultiMediaCard bus (up to 30 Cards) or with a SD Memory Card. Only one slot can be selected at a time (slots are multiplexed). A bit field in the SD Card Register performs this selection.

The SD Memory Card communication is based on a 9-pin interface (clock, command, four data and three power lines) and the MultiMedia Card on a 7-pin interface (clock, command, one data, three power lines and one reserved for future use).

The SD Memory Card interface also supports MultiMedia Card operations. The main differences between SD and MultiMedia Cards are the initialization process and the bus topology.

29.2 Block Diagram

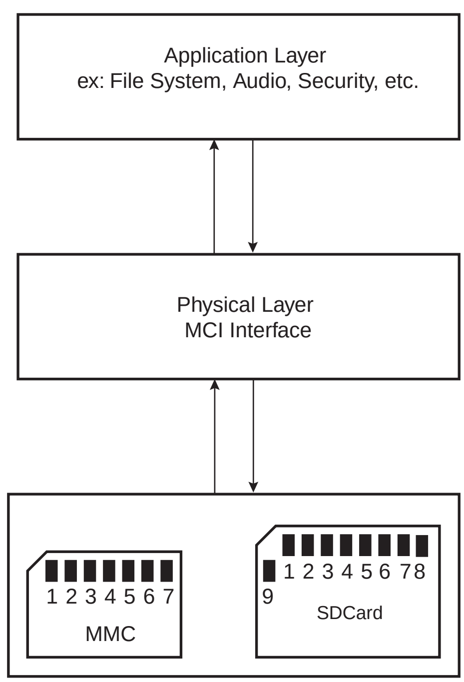
Figure 29-1. Block Diagram



Note: 1. When several MCI (x MCI) are embedded in a product, MCCK refers to MCIx_CK, MCCDA to MCIx_CDA, MCDAy to MCIx_DAy.

29.3 Application Block Diagram

Figure 29-2. Application Block Diagram



29.4 Pin Name List

Table 29-1. I/O Lines Description

Pin Name[(2)]	**Pin Description**	**Type[(1)]**	**Comments**
MCCDA	Command/response	I/O/PP/OD	CMD of an MMC or SDCard/SDIO
MCCK	Clock	I/O	CLK of an MMC or SD Card/SDIO
MCDA0 - MCDA3	Data 0..3 of Slot A	I/O/PP	DAT0 of an MMC DAT[0..3] of an SD Card/SDIO

Notes: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.
2. When several MCI (x MCI) are embedded in a product, MCCK refers to MCIx_CK, MCCDA to MCIx_CDA, MCDAy to MCIx_DAy.

29.5 Product Dependencies

29.5.1 I/O Lines

The pins used for interfacing the MultiMedia Cards or SD Cards may be multiplexed with PIO lines. The programmer must first program the PIO controllers to assign the peripheral functions to MCI pins.

29.5.2 Power Management

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

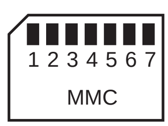
29.5.3 Interrupt

The MCI interface has an interrupt line connected to the Advanced Interrupt Controller (AIC).

Handling the MCI interrupt requires programming the AIC before configuring the MCI.

29.6 Bus Topology

Figure 29-3. Multimedia Memory Card Bus Topology



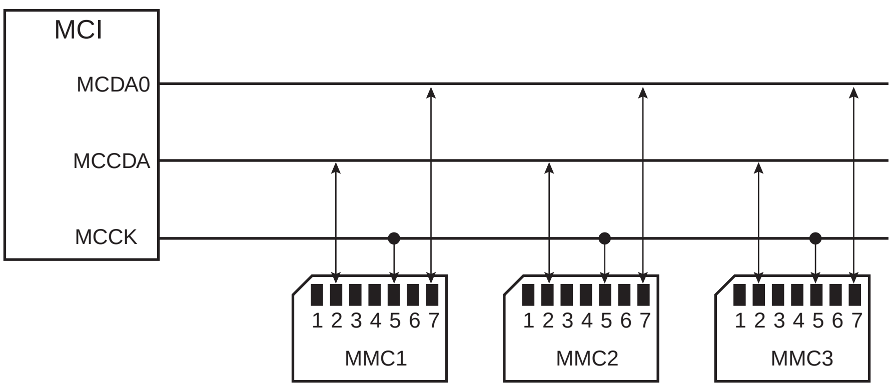
The MultiMedia Card communication is based on a 7-pin serial bus interface. It has three communication lines and four supply lines.

Table 29-2. Bus Topology

Pin Number	Name	Type(1)	Description	MCI Pin Name(2) (Slot z)
1	RSV	NC	Not connected	-
2	CMD	I/O/PP/OD	Command/response	MCCDz
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I/O	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data 0	MCDz0

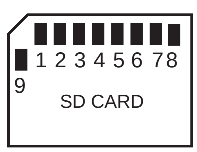
Notes: 1. I: Input, O: Output, PP: Push/Pull, OD: Open Drain.
2. When several MCI (x MCI) are embedded in a product, MCCK refers to MCIx_CK, MCCDA to MCIx_CDA, MCDAy to MCIx_DAy.

Figure 29-4. MMC Bus Connections (One Slot)



Note: When several MCI (x MCI) are embedded in a product, MCCK refers to MCIx_CK, MCCDA to MCIx_CDA MCDAy to MCIx_DAy.

Figure 29-5. SD Memory Card Bus Topology



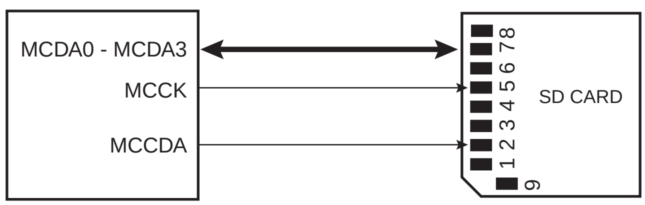
The SD Memory Card bus includes the signals listed in Table 29-3.

Table 29-3. SD Memory Card Bus Signals

Pin Number	Name	Type[(1)]	Description	MCI Pin Name[(2)] (Slot z)
1	CD/DAT[3]	I/O/PP	Card detect/ Data line Bit 3	MCDz3
2	CMD	PP	Command/response	MCCDz
3	VSS1	S	Supply voltage ground	VSS
4	VDD	S	Supply voltage	VDD
5	CLK	I/O	Clock	MCCK
6	VSS2	S	Supply voltage ground	VSS
7	DAT[0]	I/O/PP	Data line Bit 0	MCDz0
8	DAT[1]	I/O/PP	Data line Bit 1 or Interrupt	MCDz1
9	DAT[2]	I/O/PP	Data line Bit 2	MCDz2

Notes: 1. I: input, O: output, PP: Push Pull, OD: Open Drain.
2. When several MCI (x MCI) are embedded in a product, MCCK refers to MCIx_CK, MCCDA to MCIx_CDA, MCDAy to MCIx_DAy.

Figure 29-6. SD Card Bus Connections with One Slot



Note: When several MCI (x MCI) are embedded in a product, MCCK refers to MCIx_CK, MCCDA to MCIx_CDA MCDAy to MCIx_DAy.

When the MCI is configured to operate with SD memory cards, the width of the data bus can be selected in the MCI_SDCR register. Clearing the SDCBUS bit in this register means that the width is one bit; setting it means that the width is four bits. In the case of multimedia cards, only the data line 0 is used. The other data lines can be used as independent PIOs.

29.7 MultiMedia Card Operations

After a power-on reset, the cards are initialized by a special message-based MultiMedia Card bus protocol. Each message is represented by one of the following tokens:

- Command: A command is a token that starts an operation. A command is sent from the host either to a single card (addressed command) or to all connected cards (broadcast command). A command is transferred serially on the CMD line.
- Response: A response is a token which is sent from an addressed card or (synchronously) from all connected cards to the host as an answer to a previously received command. A response is transferred serially on the CMD line.
- Data: Data can be transferred from the card to the host or vice versa. Data is transferred via the data line.

Card addressing is implemented using a session address assigned during the initialization phase by the bus controller to all currently connected cards. Their unique CID number identifies individual cards.

The structure of commands, responses and data blocks is described in the MultiMedia-Card System Specification. See also Table 29-4 on page 540.

MultiMediaCard bus data transfers are composed of these tokens.

There are different types of operations. Addressed operations always contain a command and a response token. In addition, some operations have a data token; the others transfer their information directly within the command or response structure. In this case, no data token is present in an operation. The bits on the DAT and the CMD lines are transferred synchronous to the clock MCI Clock.

Two types of data transfer commands are defined:

- Sequential commands: These commands initiate a continuous data stream. They are terminated only when a stop command follows on the CMD line. This mode reduces the command overhead to an absolute minimum.
- Block-oriented commands: These commands send a data block succeeded by CRC bits.

Both read and write operations allow either single or multiple block transmission. A multiple block transmission is terminated when a stop command follows on the CMD line similarly to the sequential read or when a multiple block transmission has a pre-defined block count (See "Data Transfer Operation" on page 541.).

The MCI provides a set of registers to perform the entire range of MultiMedia Card operations.

29.7.1 Command - Response Operation

After reset, the MCI is disabled and becomes valid after setting the MCIEN bit in the MCI_CR Control Register.

The PWSEN bit saves power by dividing the MCI clock by $2^{PWSDIV} + 1$ when the bus is inactive.

The two bits, RDPROOF and WRPROOF in the MCI Mode Register (MCI_MR) allow stopping the MCI Clock during read or write access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

The command and the response of the card are clocked out with the rising edge of the MCI Clock.

All the timings for MultiMedia Card are defined in the MultiMediaCard System Specification.

The two bus modes (open drain and push/pull) needed to process all the operations are defined in the MCI command register. The MCI_CMDR allows a command to be carried out.

For example, to perform an ALL_SEND_CID command:

	Host Command					N_{ID} Cycles					CID			
CMD	S	T	Content	CRC	E	Z	******	Z	S	T	Content	Z	Z	Z

The command ALL_SEND_CID and the fields and values for the MCI_CMDR Control Register are described in Table 29-4 and Table 29-5.

Table 29-4. 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

Note: bcr means broadcast command with response.

Table 29-5. Fields and Values for MCI_CMDR Command Register

Field	Value
CMDNB (command number)	2 (CMD2)
RSPTYP (response type)	2 (R2: 136 bits response)
SPCMD (special command)	0 (not a special command)
OPCMD (open drain command)	1
MAXLAT (max latency for command to response)	0 (NID cycles ==> 5 cycles)
TRCMD (transfer command)	0 (No transfer)
TRDIR (transfer direction)	X (available only in transfer command)
TRTYP (transfer type)	X (available only in transfer command)
IOSPCMD (SDIO special command)	0 (not a special command)

The MCI_ARGR contains the argument field of the command.

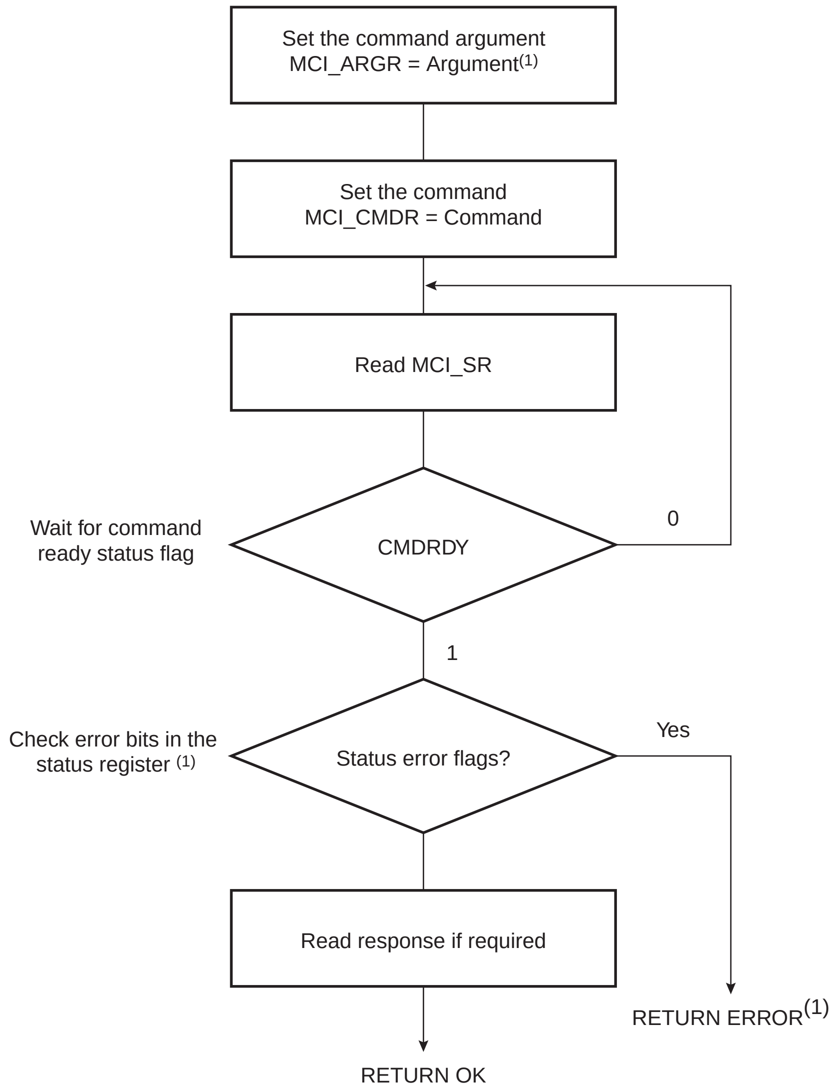
To send a command, the user must perform the following steps:

- Fill the argument register (MCI_ARGR) with the command argument.
- Set the command register (MCI_CMDR) (see Table 29-5).

The command is sent immediately after writing the command register. The status bit CMDRDY in the status register (MCI_SR) is asserted when the command is completed. If the command requires a response, it can be read in the MCI response register (MCI_RSPR). The response size can be from 48 bits up to 136 bits depending on the command. The MCI embeds an error detection to prevent any corrupted data during the transfer.

The following flowchart shows how to send a command to the card and read the response if needed. In this example, the status register bits are polled but setting the appropriate bits in the interrupt enable register (MCI_IER) allows using an interrupt method.

Figure 29-7. Command/Response Functional Flow Diagram



Note: 1. If the command is SEND_OP_COND, the CRC error flag is always present (refer to R3 response in the MultiMedia Card specification).

29.7.2 Data Transfer Operation

The MultiMedia Card allows several read/write operations (single block, multiple blocks, stream, etc.). These kind of transfers can be selected setting the Transfer Type (TRTYP) field in the MCI Command Register (MCI_CMDR).

These operations can be done using the features of the Peripheral DMA Controller (PDC). If the PDCMODE bit is set in MCI_MR, then all reads and writes use the PDC facilities.

In all cases, the block length (BLKLEN field) must be defined either in the mode register MCI_MR, or in the Block Register MCI_BLKR. This field determines the size of the data block.

Enabling PDC Force Byte Transfer (PDCFBYTE bit in the MCI_MR) allows the PDC to manage with internal byte transfers, so that transfer of blocks with a size different from modulo 4 can be supported. When PDC Force Byte Transfer is disabled, the PDC type of transfers are in words, otherwise the type of transfers are in bytes.

Consequent to MMC Specification 3.1, two types of multiple block read (or write) transactions are defined (the host can use either one at any time):

- Open-ended/Infinite Multiple block read (or write):

  The number of blocks for the read (or write) multiple block operation is not defined. The card will continuously transfer (or program) data blocks until a stop transmission command is received.

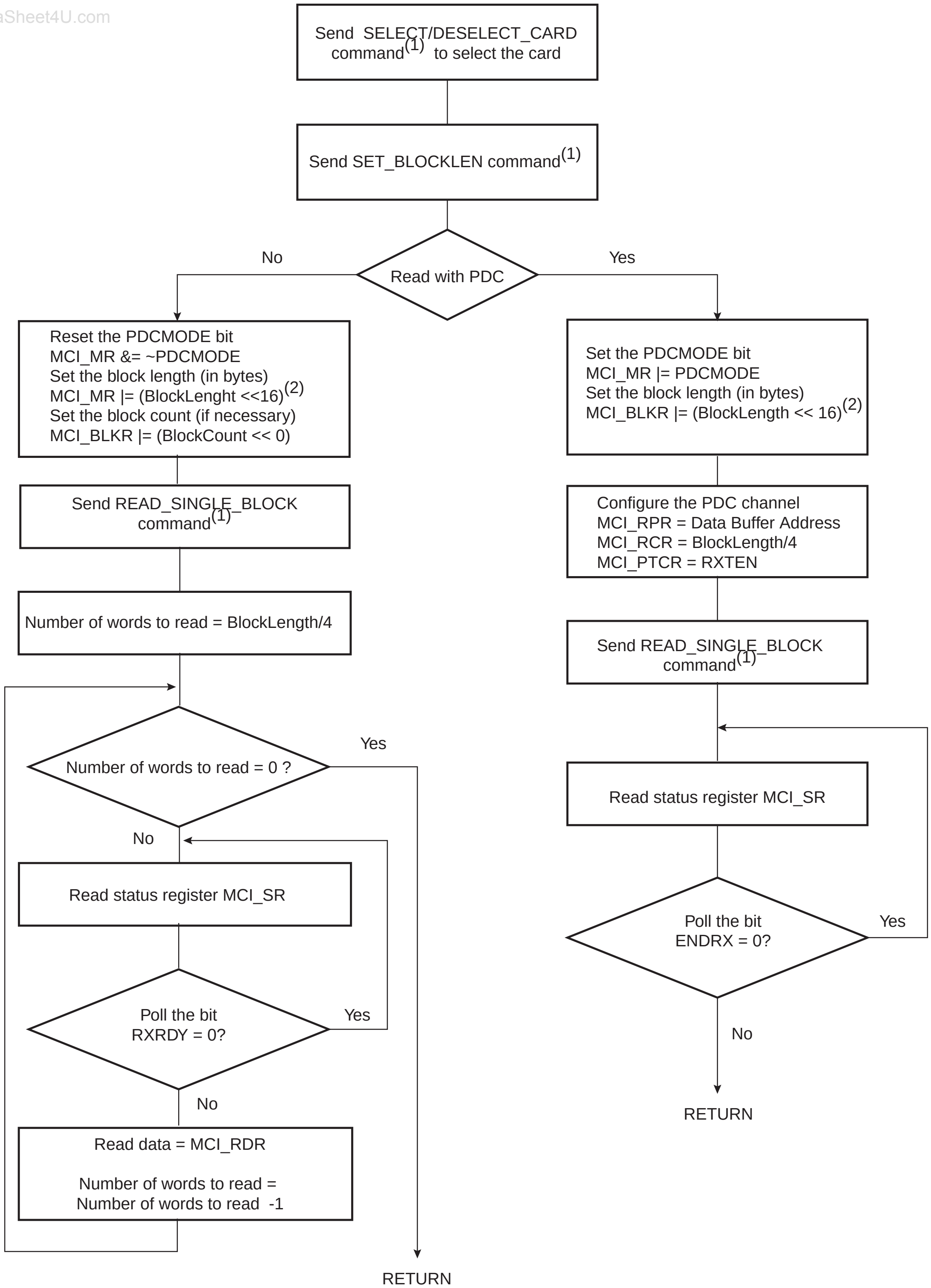
- Multiple block read (or write) with pre-defined block count (since version 3.1 and higher):

  The card will transfer (or program) the requested number of data blocks and terminate the transaction. The stop command is not required at the end of this type of multiple block read (or write), unless terminated with an error. In order to start a multiple block read (or write) with pre-defined block count, the host must correctly program the MCI Block Register (MCI_BLKR). Otherwise the card will start an open-ended multiple block read. The BCNT field of the Block Register defines the number of blocks to transfer (from 1 to 65535 blocks). Programming the value 0 in the BCNT field corresponds to an infinite block transfer.

29.7.3 Read Operation

The following flowchart shows how to read a single block with or without use of PDC facilities. In this example (see Figure 29-8), 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.

Figure 29-8. Read Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 29-7).
2. This field is also accessible in the MCI Block Register (MCI_BLKR).

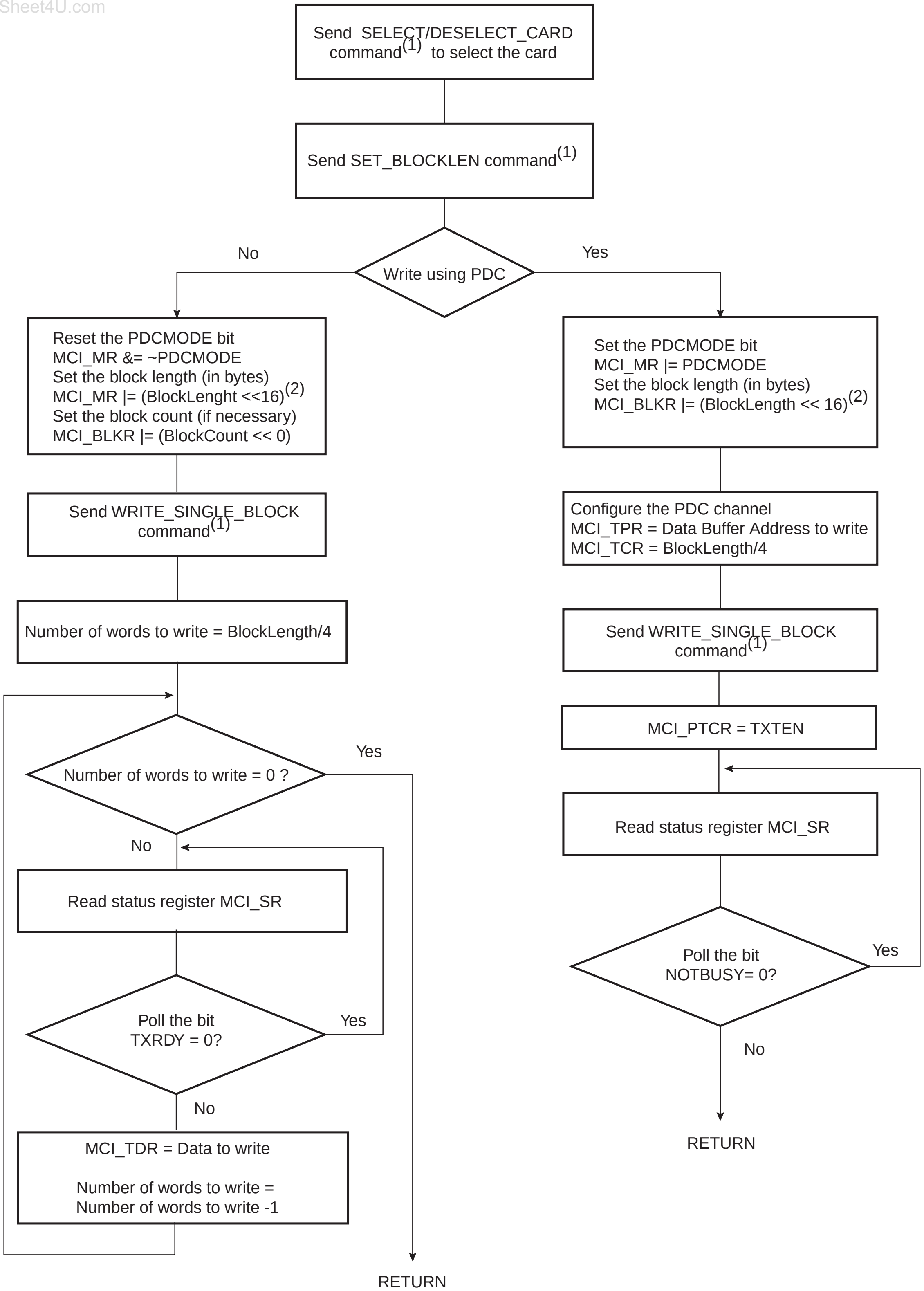
29.7.4 Write Operation

In write operation, the MCI Mode Register (MCI_MR) is used to define the padding value when writing non-multiple block size. If the bit PDCPADV is 0, then 0x00 value is used when padding data, otherwise 0xFF is used.

If set, the bit PDCMODE enables PDC transfer.

The following flowchart shows how to write a single block with or without use of PDC facilities (see Figure 29-9). Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (MCI_IMR).

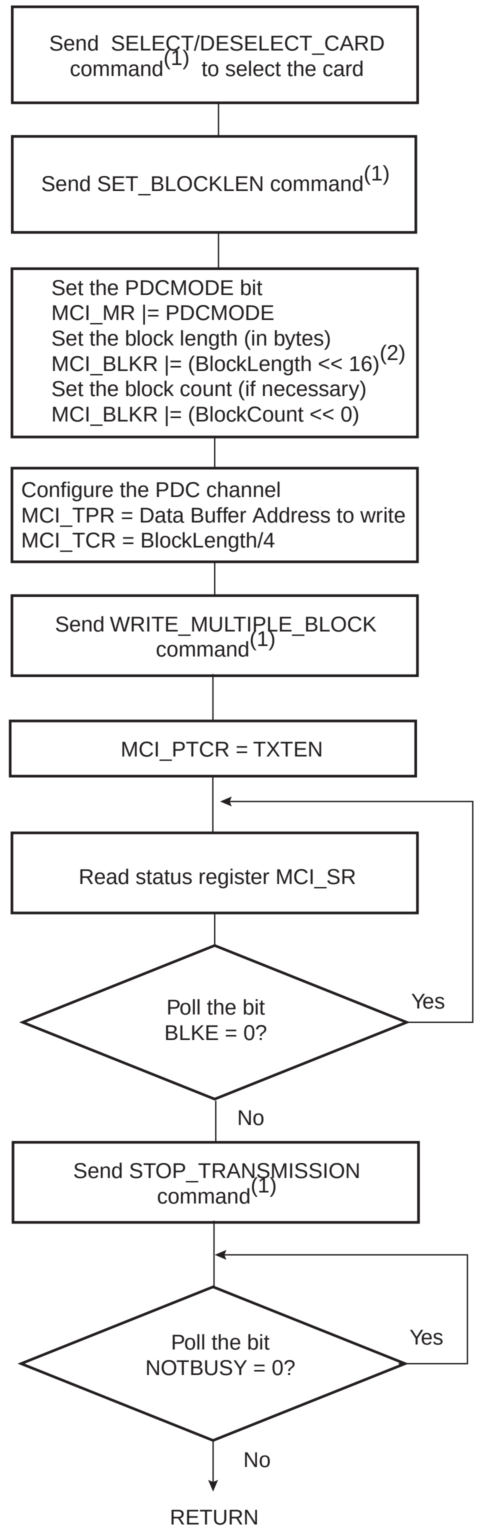
Figure 29-9. Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 29-7).
2. This field is also accessible in the MCI Block Register (MCI_BLKR).

The following flowchart shows how to manage a multiple write block transfer with the PDC (see Figure 29-10). Polling or interrupt method can be used to wait for the end of write according to the contents of the Interrupt Mask Register (MCI_IMR).

Figure 29-10. Multiple Write Functional Flow Diagram



Note: 1. It is assumed that this command has been correctly sent (see Figure 29-7).

2. This field is also accessible in the MCI Block Register (MCI_BLKR).

29.8 SD/SDIO Card Operations

The MultiMedia Card Interface allows processing of SD Memory (Secure Digital Memory Card) and SDIO (SD Input Output) Card commands.

SD/SDIO cards are based on the Multi Media Card (MMC) format, but are physically slightly thicker and feature higher data transfer rates, a lock switch on the side to prevent accidental overwriting and security features. The physical form factor, pin assignment and data transfer protocol are forward-compatible with the MultiMedia Card with some additions. SD slots can actually be used for more than flash memory cards. Devices that support SDIO can use small devices designed for the SD form factor, such as GPS receivers, Wi-Fi or Bluetooth adapters, modems, barcode readers, IrDA adapters, FM radio tuners, RFID readers, digital cameras and more.

SD/SDIO is covered by numerous patents and trademarks, and licensing is only available through the Secure Digital Card Association.

The SD/SDIO 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/SDIO Card and the MultiMedia Card is the initialization process.

The SD/SDIO Card Register (MCI_SDCR) allows selection of the Card Slot and the data bus width.

The SD/SDIO Card bus allows dynamic configuration of the number of data lines. After power up, by default, the SD/SDIO Card uses only DAT0 for data transfer. After initialization, the host can change the bus width (number of active data lines).

29.8.1 SDIO Data Transfer Type

SDIO cards may transfer data in either a multi-byte (1 to 512 bytes) or an optional block format (1 to 511 blocks), while the SD memory cards are fixed in the block transfer mode. The TRTYP field in the MCI Command Register (MCI_CMDR) allows to choose between SDIO Byte or SDIO Block transfer.

The number of bytes/blocks to transfer is set through the BCNT field in the MCI Block Register (MCI_BLKR). In SDIO Block mode, the field BLKLEN must be set to the data block size while this field is not used in SDIO Byte mode.

An SDIO Card can have multiple I/O or combined I/O and memory (called Combo Card). Within a multi-function SDIO or a Combo card, there are multiple devices (I/O and memory) that share access to the SD bus. In order to allow the sharing of access to the host among multiple devices, SDIO and combo cards can implement the optional concept of suspend/resume (Refer to the SDIO Specification for more details). To send a suspend or a resume command, the host must set the SDIO Special Command field (IOSPCMD) in the MCI Command Register.

29.8.2 SDIO Interrupts

Each function within an SDIO or Combo card may implement interrupts (Refer to the SDIO Specification for more details). In order to allow the SDIO card to interrupt the host, an interrupt function is added to a pin on the DAT[1] line to signal the card's interrupt to the host. An SDIO interrupt on each slot can be enabled through the MCI Interrupt Enable Register. The SDIO interrupt is sampled regardless of the currently selected slot.

29.9 MultiMedia Card Interface (MCI) User Interface

Table 29-6. Register Mapping

Offset	Register	Register Name	Read/Write	Reset
0x00	Control Register	MCI_CR	Write	–
0x04	Mode Register	MCI_MR	Read/write	0x0
0x08	Data Timeout Register	MCI_DTOR	Read/write	0x0
0x0C	SD/SDIO Card Register	MCI_SDCR	Read/write	0x0
0x10	Argument Register	MCI_ARGR	Read/write	0x0
0x14	Command Register	MCI_CMDR	Write	–
0x18	Block Register	MCI_BLKR	Read/write	0x0
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
0x50-0xFC	Reserved	–	–	–
0x100-0x124	Reserved for the PDC	–	–	–

Note: 1. The response register can be read by N accesses at the same MCI_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

29.9.1 MCI Control Register

Name: MCI_CR

Access Type: Write-only

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

- **MCIEN: Multi-Media Interface Enable**

0 = No effect.

1 = Enables the Multi-Media Interface if MCDIS is 0.

- **MCIDIS: Multi-Media Interface Disable**

0 = No effect.

1 = Disables the Multi-Media Interface.

- **PWSEN: Power Save Mode Enable**

0 = No effect.

1 = Enables the Power Saving Mode if PWSDIS is 0.

Warning: Before enabling this mode, the user must set a value different from 0 in the PWSDIV field (Mode Register MCI_MR).

- **PWSDIS: Power Save Mode Disable**

0 = No effect.

1 = Disables the Power Saving Mode.

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the MCI. A software triggered hardware reset of the MCI interface is performed.

29.9.2 MCI Mode Register

Name: MCI_MR

Access Type: Read/write

31	30	29	28	27	26	25	24
BLKLEN							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
BLKLEN							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
PDCMODE	PDCPADV	PDCFBYTE	WRPROOF	RDPROOF	PWSDIV		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CLKDIV							

- **CLKDIV: Clock Divider**

Multimedia Card Interface clock (MCCK or MCI_CK) is Master Clock (MCK) divided by (2*(CLKDIV+1)).

- **PWSDIV: Power Saving Divider**

Multimedia Card Interface clock is divided by $2^{(PWSDIV)}$ + 1 when entering Power Saving Mode.

Warning: This value must be different from 0 before enabling the Power Save Mode in the MCI_CR (MCI_PWSEN bit).

- **RDPROOF Read Proof Enable**

Enabling Read Proof allows to stop the MCI Clock during read access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

0 = Disables Read Proof.

1 = Enables Read Proof.

- **WRPROOF Write Proof Enable**

Enabling Write Proof allows to stop the MCI Clock during write access if the internal FIFO is full. This will guarantee data integrity, not bandwidth.

0 = Disables Write Proof.

1 = Enables Write Proof.

- **PDCFBYTE: PDC Force Byte Transfer**

Enabling PDC Force Byte Transfer allows the PDC to manage with internal byte transfers, so that transfer of blocks with a size different from modulo 4 can be supported.

Warning: BLKLEN value depends on PDCFBYTE.

0 = Disables PDC Force Byte Transfer. PDC type of transfer are in words.

1 = Enables PDC Force Byte Transfer. PDC type of transfer are in bytes.

- **PDCPADV: PDC Padding Value**

0 = 0x00 value is used when padding data in write transfer (not only PDC transfer).

1 = 0xFF value is used when padding data in write transfer (not only PDC transfer).

- **PDCMODE: PDC-oriented Mode**

0 = Disables PDC transfer

1 = Enables PDC transfer. In this case, UNRE and OVRE flags in the MCI Mode Register (MCI_SR) are deactivated after the PDC transfer has been completed.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

This field is also accessible in the MCI Block Register (MCI_BLKR).

Bits 16 and 17 must be set to 0 if PDCFBYTE is disabled.

Note: In SDIO Byte mode, BLKLEN field is not used.

29.9.3 MCI Data Timeout Register

Name: MCI_DTOR

Access Type: Read/write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	DTOMUL			DTOCYC			

- **DTOCYC: Data Timeout Cycle Number**
- **DTOMUL: Data Timeout Multiplier**

These fields determine the maximum number of Master Clock cycles that the MCI waits between two data block transfers. It equals (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

If the data time-out set by DTOCYC and DTOMUL has been exceeded, the Data Time-out Error flag (DTOE) in the MCI Status Register (MCI_SR) raises.

29.9.4 MCI SDCard/SDIO Register

Name: MCI_SDCR

Access Type: Read/write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
SDCBUS	–	–	–	–	–	SDCSEL	

- **SDCSEL: SDCard/SDIO Slot**

SDCSEL		SDCard/SDIO Slot
0	0	Slot A is selected.
0	1	Reserved
1	0	Reserved
1	1	Reserved

- **SDCBUS: SDCard/SDIO Bus Width**

0 = 1-bit data bus

1 = 4-bit data bus

29.9.5 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**

29.9.6 MCI Command Register

Name: MCI_CMDR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	IOSPCMD	
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	TRTYP			TRDIR	TRCMD	
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	MAXLAT	OPDCMD	SPCMD		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RSPTYP		CMDNB					

This register is write-protected while CMDRDY is 0 in MCI_SR. If an Interrupt command is sent, this register is only writable by an interrupt response (field SPCMD). This means that the current command execution cannot be interrupted or modified.

- **CMDNB: Command Number**
- **RSPTYP: Response Type**

RSP		Response Type
0	0	No response.
0	1	48-bit response.
1	0	136-bit response.
1	1	Reserved.

- **SPCMD: Special Command**

SPCMD			Command
0	0	0	Not a special CMD.
0	0	1	Initialization CMD: 74 clock cycles for initialization sequence.
0	1	0	Synchronized CMD: Wait for the end of the current data block transfer before sending the pending command.
0	1	1	Reserved.
1	0	0	Interrupt command: Corresponds to the Interrupt Mode (CMD40).
1	0	1	Interrupt response: Corresponds to the Interrupt Mode (CMD40).

- **OPDCMD: Open Drain Command**

0 = Push pull command

1 = Open drain command

- **MAXLAT: Max Latency for Command to Response**

0 = 5-cycle max latency

1 = 64-cycle max latency

- **TRCMD: Transfer Command**

TRCMD		Transfer Type
0	0	No data transfer
0	1	Start data transfer
1	0	Stop data transfer
1	1	Reserved

- **TRDIR: Transfer Direction**

0 = Write

1 = Read

- **TRTYP: Transfer Type**

TRTYP			Transfer Type
0	0	0	MMC/SDCard Single Block
0	0	1	MMC/SDCard Multiple Block
0	1	0	MMC Stream
0	1	1	Reserved
1	0	0	SDIO Byte
1	0	1	SDIO Block
1	1	0	Reserved
1	1	1	Reserved

- **IOSPCMD: SDIO Special Command**

IOSPCMD		SDIO Special Command Type
0	0	Not a SDIO Special Command
0	1	SDIO Suspend Command
1	0	SDIO Resume Command
1	1	Reserved

29.9.7 MCI Block Register

Name: MCI_BLKR

Access Type: Read/write

31	30	29	28	27	26	25	24
BLKLEN							
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
BLKLEN							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**

BCNT							

7	6	5	4	3	2	1	0
BCNT							

- **BCNT: MMC/SDIO Block Count - SDIO Byte Count**

This field determines the number of data byte(s) or block(s) to transfer.

The transfer data type and the authorized values for BCNT field are determined by the TRTYP field in the MCI Command Register (MCI_CMDR):

TRTYP			Type of Transfer	BCNT Authorized Values
0	0	1	MMC/SDCard Multiple Block	From 1 to MCI_MAXNUM_BLK: Value 0 corresponds to an infinite block transfer.
1	0	0	SDIO Byte	From 1 to 512 bytes: Value 0 corresponds to a 512-byte transfer. Values from 0x200 to 0xFFFF are forbidden.
1	0	1	SDIO Block	From 1 to 511 blocks: Value 0 corresponds to an infinite block transfer. Values from 0x200 to 0xFFFF are forbidden.
Other values			-	Reserved.

Warning: In SDIO Byte and Block modes, writing to the 7 last bits of BCNT field, is forbidden and may lead to unpredictable results.

- **BLKLEN: Data Block Length**

This field determines the size of the data block.

This field is also accessible in the MCI Mode Register (MCI_MR).

Bits 16 and 17 must be set to 0 if PDCFBYTE is disabled.

Note: In SDIO Byte mode, BLKLEN field is not used.

29.9.8 MCI Response Register

Name: MCI_RSPR

Access Type: Read-only

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

- **RSP: Response**

Note: 1. The response register can be read by N accesses at the same MCI_RSPR or at consecutive addresses (0x20 to 0x2C). N depends on the size of the response.

29.9.9 MCI Receive Data Register

Name: MCI_RDR

Access Type: Read-only

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

- **DATA: Data to Read**

29.9.10 MCI Transmit Data Register

Name: MCI_TDR

Access Type: Write-only

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

- **DATA: Data to Write**

29.9.11 MCI Status Register

Name: MCI_SR

Access Type: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
TXBUFE	RXBUFF	–	–	-	-	-	SDIOIRQA
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ENDTX	ENDRX	NOTBUSY	DTIP	BLKE	TXRDY	RXRDY	CMDRDY

- **CMDRDY: Command Ready**

0 = A command is in progress.

1 = The last command has been sent. Cleared when writing in the MCI_CMDR.

- **RXRDY: Receiver Ready**

0 = Data has not yet been received since the last read of MCI_RDR.

1 = Data has been received since the last read of MCI_RDR.

- **TXRDY: Transmit Ready**

0= The last data written in MCI_TDR has not yet been transferred in the Shift Register.

1= The last data written in MCI_TDR has been transferred in the Shift Register.

- **BLKE: Data Block Ended**

This flag must be used only for Write Operations.

0 = A data block transfer is not yet finished. Cleared when reading the MCI_SR.

1 = A data block transfer has ended, including the CRC16 Status transmission.
In PDC mode (PDCMODE=1), the flag is set when the CRC Status of the last block has been transmitted (TXBUFE already set).
Otherwise (PDCMODE=0), the flag is set for each transmitted CRC Status.

Refer to the MMC or SD Specification for more details concerning the CRC Status.

- **DTIP: Data Transfer in Progress**

0 = No data transfer in progress.

1 = The current data transfer is still in progress, including CRC16 calculation. Cleared at the end of the CRC16 calculation.

- **NOTBUSY: MCI Not Busy**

This flag must be used only for Write Operations.

A block write operation uses a simple busy signalling of the write operation duration on the data (DAT0) line: during a data transfer block, if the card does not have a free data receive buffer, the card indicates this condition by pulling down the data line (DAT0) to LOW. The card stops pulling down the data line as soon as at least one receive buffer for the defined data transfer block length becomes free.

The NOTBUSY flag allows to deal with these different states.

0 = The MCI is not ready for new data transfer. Cleared at the end of the card response.

1 = The MCI is ready for new data transfer. Set when the busy state on the data line has ended. This corresponds to a free internal data receive buffer of the card.

Refer to the MMC or SD Specification for more details concerning the busy behavior.

- **ENDRX: End of RX Buffer**

0 = The Receive Counter Register has not reached 0 since the last write in MCI_RCR or MCI_RNCR.

1 = The Receive Counter Register has reached 0 since the last write in MCI_RCR or MCI_RNCR.

- **ENDTX: End of TX Buffer**

0 = The Transmit Counter Register has not reached 0 since the last write in MCI_TCR or MCI_TNCR.

1 = The Transmit Counter Register has reached 0 since the last write in MCI_TCR or MCI_TNCR.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the MCI Controller.

- **RXBUFF: RX Buffer Full**

0 = MCI_RCR or MCI_RNCR has a value other than 0.

1 = Both MCI_RCR and MCI_RNCR have a value of 0.

- **TXBUFE: TX Buffer Empty**

0 = MCI_TCR or MCI_TNCR has a value other than 0.

1 = Both MCI_TCR and MCI_TNCR have a value of 0.

Note: BLKE and NOTBUSY flags can be used to check that the data has been successfully transmitted on the data lines and not only transferred from the PDC to the MCI Controller.

- **RINDE: Response Index Error**

0 = No error.

1 = A mismatch is detected between the command index sent and the response index received. Cleared when writing in the MCI_CMDR.

- **RDIRE: Response Direction Error**

0 = No error.

1 = The direction bit from card to host in the response has not been detected.

- **RCRCE: Response CRC Error**

0 = No error.

1 = A CRC7 error has been detected in the response. Cleared when writing in the MCI_CMDR.

- **RENDE: Response End Bit Error**

0 = No error.

1 = The end bit of the response has not been detected. Cleared when writing in the MCI_CMDR.

- **RTOE: Response Time-out Error**

0 = No error.

1 = The response time-out set by MAXLAT in the MCI_CMDR has been exceeded. Cleared when writing in the MCI_CMDR.

- **DCRCE: Data CRC Error**

0 = No error.

1 = A CRC16 error has been detected in the last data block. Cleared by reading in the MCI_SR register.

- **DTOE: Data Time-out Error**

0 = No error.

1 = The data time-out set by DTOCYC and DTOMUL in MCI_DTOR has been exceeded. Cleared by reading in the MCI_SR register.

- **OVRE: Overrun**

0 = No error.

1 = At least one 8-bit received data has been lost (not read). Cleared when sending a new data transfer command.

- **UNRE: Underrun**

0 = No error.

1 = At least one 8-bit data has been sent without valid information (not written). Cleared when sending a new data transfer command.

- **SDIOIRQA: SDIO Interrupt for Slot A**

0 = No interrupt detected on SDIO Slot A.

1 = A SDIO Interrupt on Slot A has reached. Cleared when reading the MCI_SR.

- **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.

29.9.12 MCI Interrupt Enable Register

Name: MCI_IER

Access Type: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
TXBUFE	RXBUFF	–	–	-	-	-	SDIOIRQA
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**

- **SDIOIRQA: SDIO Interrupt for Slot A 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.

29.9.13 MCI Interrupt Disable Register

Name: MCI_IDR

Access Type: Write-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
TXBUFE	RXBUFF	–	–	-	-	-	SDIOIRQA
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**
- **SDIOIRQA: SDIO Interrupt for Slot A 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.

29.9.14 MCI Interrupt Mask Register

Name: MCI_IMR

Access Type: Read-only

31	30	29	28	27	26	25	24
UNRE	OVRE	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	DTOE	DCRCE	RTOE	RENDE	RCRCE	RDIRE	RINDE
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
TXBUFE	RXBUFF	–	–	-	-	-	SDIOIRQA
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**
- **SDIOIRQA: SDIO Interrupt for Slot A 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.

30. USB Host Port (UHP)

30.1 Description

The USB Host Port (UHP) interfaces the USB with the host application. It handles Open HCI protocol (Open Host Controller Interface) as well as USB v2.0 Full-speed and Low-speed protocols.

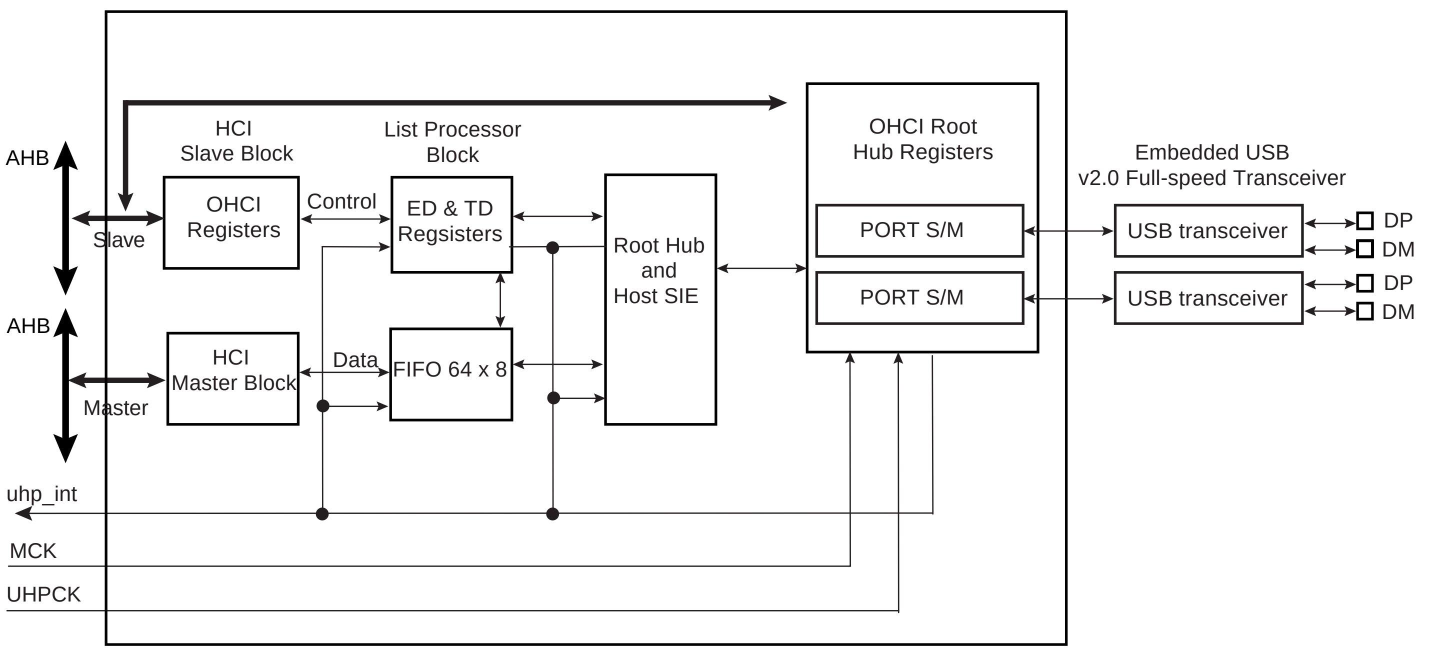
The USB Host Port integrates a root hub and transceivers on downstream ports. It provides several high-speed half-duplex serial communication ports at a baud rate of 12 Mbit/s. Up to 127 USB devices (printer, camera, mouse, keyboard, disk, etc.) and the USB hub can be connected to the USB host in the USB “tiered star” topology.

The USB Host Port controller is fully compliant with the OpenHCI specification. The USB Host Port User Interface (registers description) can be found in the Open HCI Rev 1.0 Specification available on http://h18000.www1.hp.com/productinfo/development/openhci.html. The standard OHCI USB stack driver can be easily ported to ATMEL’s architecture in the same way all existing class drivers run without hardware specialization.

This means that all standard class devices are automatically detected and available to the user application. As an example, integrating an HID (Human Interface Device) class driver provides a plug & play feature for all USB keyboards and mouses.

30.2 Block Diagram

Figure 30-1. Block Diagram



Access to the USB host operational registers is achieved through the AHB bus slave interface. The OpenHCI host controller initializes master DMA transfers through the AHB bus master interface as follows:

- Fetches endpoint descriptors and transfer descriptors
- Access to endpoint data from system memory

- Access to the HC communication area
- Write status and retire transfer Descriptor

Memory access errors (abort, misalignment) lead to an “UnrecoverableError” indicated by the corresponding flag in the host controller operational registers.

The USB root hub is integrated in the USB host. Several USB downstream ports are available. The number of downstream ports can be determined by the software driver reading the root hub’s operational registers. Device connection is automatically detected by the USB host port logic.

Warning: A pull-down must be connected to DP on the board. Otherwise the USB host will permanently detect a device connection on this port.

USB physical transceivers are integrated in the product and driven by the root hub’s ports.

Over current protection on ports can be activated by the USB host controller. Atmel’s standard product does not dedicate pads to external over current protection.

30.3 Product Dependencies

– I/O Lines

DPs and DMs are not controlled by any PIO controllers. The embedded USB physical transceivers are controlled by the USB host controller.

30.3.1 Power Management

The USB host controller requires a 48 MHz clock. This clock must be generated by a PLL with a correct accuracy of ± 0.25%.

Thus the USB device peripheral receives two clocks from the Power Management Controller (PMC): the master clock MCK used to drive the peripheral user interface (MCK domain) and the UHPCLK 48 MHz clock used to interface with the bus USB signals (Recovered 12 MHz domain).

30.3.2 Interrupt

The USB host interface has an interrupt line connected to the Advanced Interrupt Controller (AIC).

Handling USB host interrupts requires programming the AIC before configuring the UHP.

30.4 Functional Description

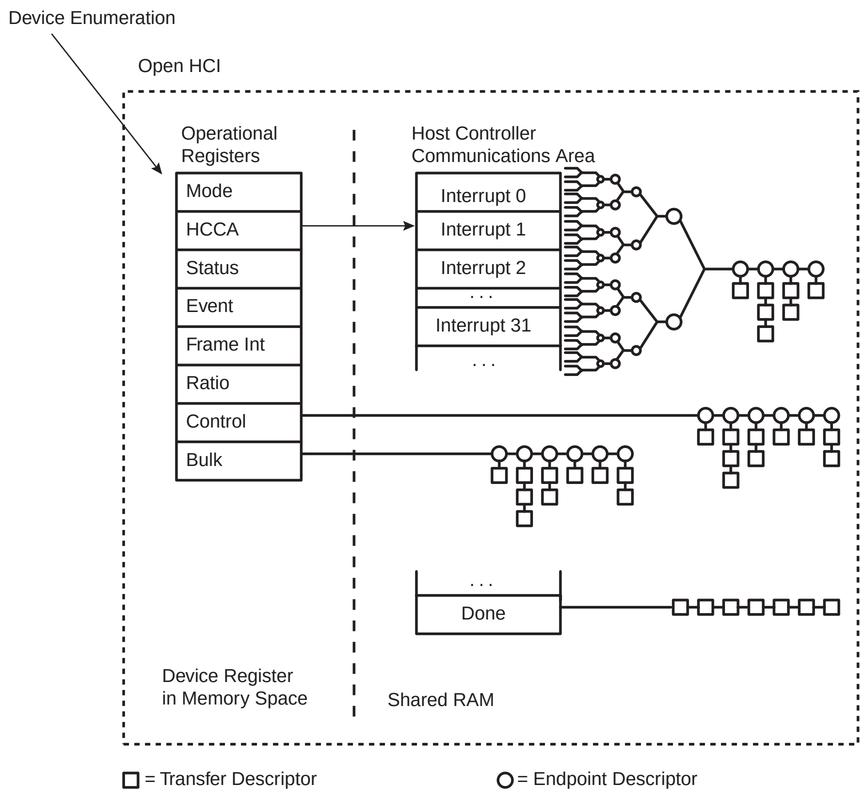
Please refer to the Open Host Controller Interface Specification for USB Release 1.0.a.

30.4.1 Host Controller Interface

There are two communication channels between the Host Controller and the Host Controller Driver. The first channel uses a set of operational registers located on the USB Host Controller. The Host Controller is the target for all communications on this channel. The operational registers contain control, status and list pointer registers. They are mapped in the memory mapped area. Within the operational register set there is a pointer to a location in the processor address space named the Host Controller Communication Area (HCCA). The HCCA is the second communication channel. The host controller is the master for all communication on this channel. The HCCA contains the head pointers to the interrupt Endpoint Descriptor lists, the head pointer to the done queue and status information associated with start-of-frame processing.

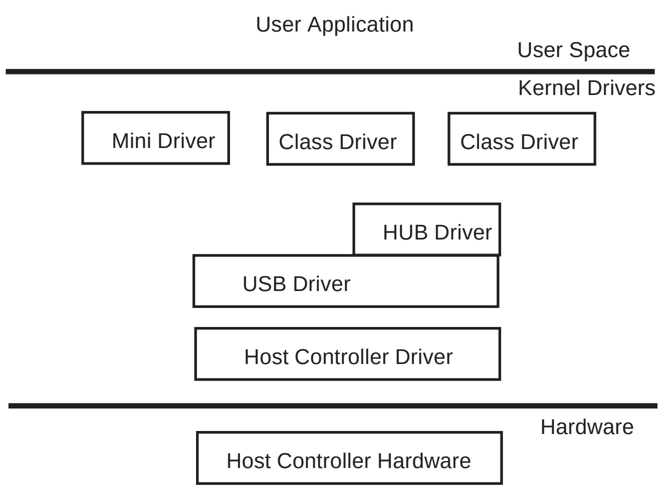
The basic building blocks for communication across the interface are Endpoint Descriptors (ED, 4 double words) and Transfer Descriptors (TD, 4 or 8 double words). The host controller assigns an Endpoint Descriptor to each endpoint in the system. A queue of Transfer Descriptors is linked to the Endpoint Descriptor for the specific endpoint.

Figure 30-2. USB Host Communication Channels



30.4.2 Host Controller Driver

Figure 30-3. USB Host Drivers

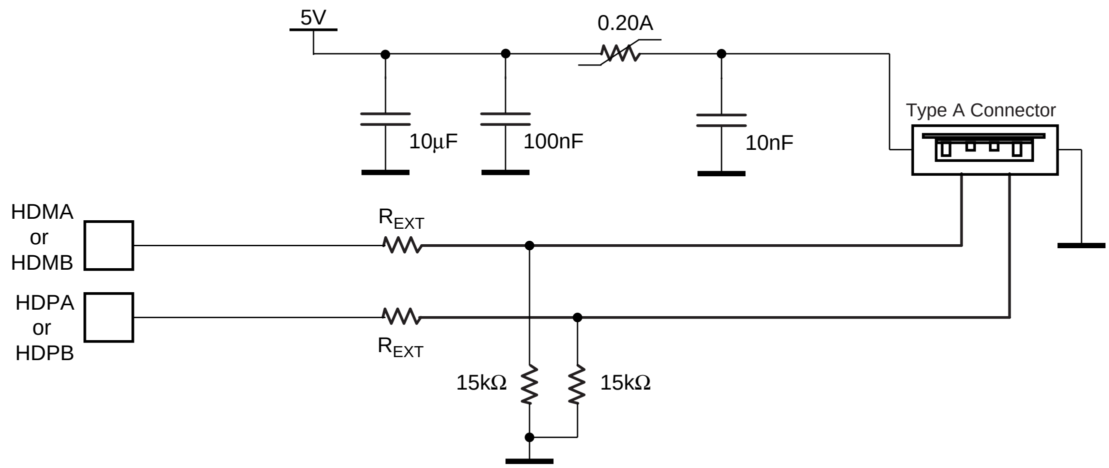


USB Handling is done through several layers as follows:

- Host controller hardware and serial engine: Transmits and receives USB data on the bus.
- Host controller driver: Drives the Host controller hardware and handles the USB protocol.
- USB Bus driver and hub driver: Handles USB commands and enumeration. Offers a hardware independent interface.
- Mini driver: Handles device specific commands.
- Class driver: Handles standard devices. This acts as a generic driver for a class of devices, for example the HID driver.

30.5 Typical Connection

Figure 30-4. Board Schematic to Interface UHP Device Controller



As device connection is automatically detected by the USB host port logic, a pull-down must be connected on DP and DM on the board. Otherwise the USB host permanently detects a device connection on this port.

A termination serial resistor must be connected to HDP and HDM. The resistor value is defined in the electrical specification of the product (R_{EXT}).

31. USB Device Port (UDP)

31.1 Description

The USB Device Port (UDP) is compliant with the Universal Serial Bus (USB) V2.0 full-speed device specification.

Each endpoint can be configured in one of several USB transfer types. It can be associated with one or two banks of a dual-port RAM used to store the current data payload. If two banks are used, one DPR bank is read or written by the processor, while the other is read or written by the USB device peripheral. This feature is mandatory for isochronous endpoints. Thus the device maintains the maximum bandwidth (1M bytes/s) by working with endpoints with two banks of DPR.

Table 31-1. USB Endpoint Description

Endpoint Number	Mnemonic	Dual-Bank	Max. Endpoint Size	Endpoint Type
0	EP0	No	64	Control/Bulk/Interrupt
1	EP1	Yes	64	Bulk/Iso/Interrupt
2	EP2	Yes	64	Bulk/Iso/Interrupt
3	EP3	No	64	Control/Bulk/Interrupt
4	EP4	Yes	512	Bulk/Iso/Interrupt
5	EP5	Yes	1023	Bulk/Iso/Interrupt
6	EP6	No	64	Bulk/Interrupt
7	EP7	No	64	Bulk/Interrupt

Suspend and resume are automatically detected by the USB device, which notifies the processor by raising an interrupt. Depending on the product, an external signal can be used to send a wake up to the USB host controller.

31.2 Block Diagram

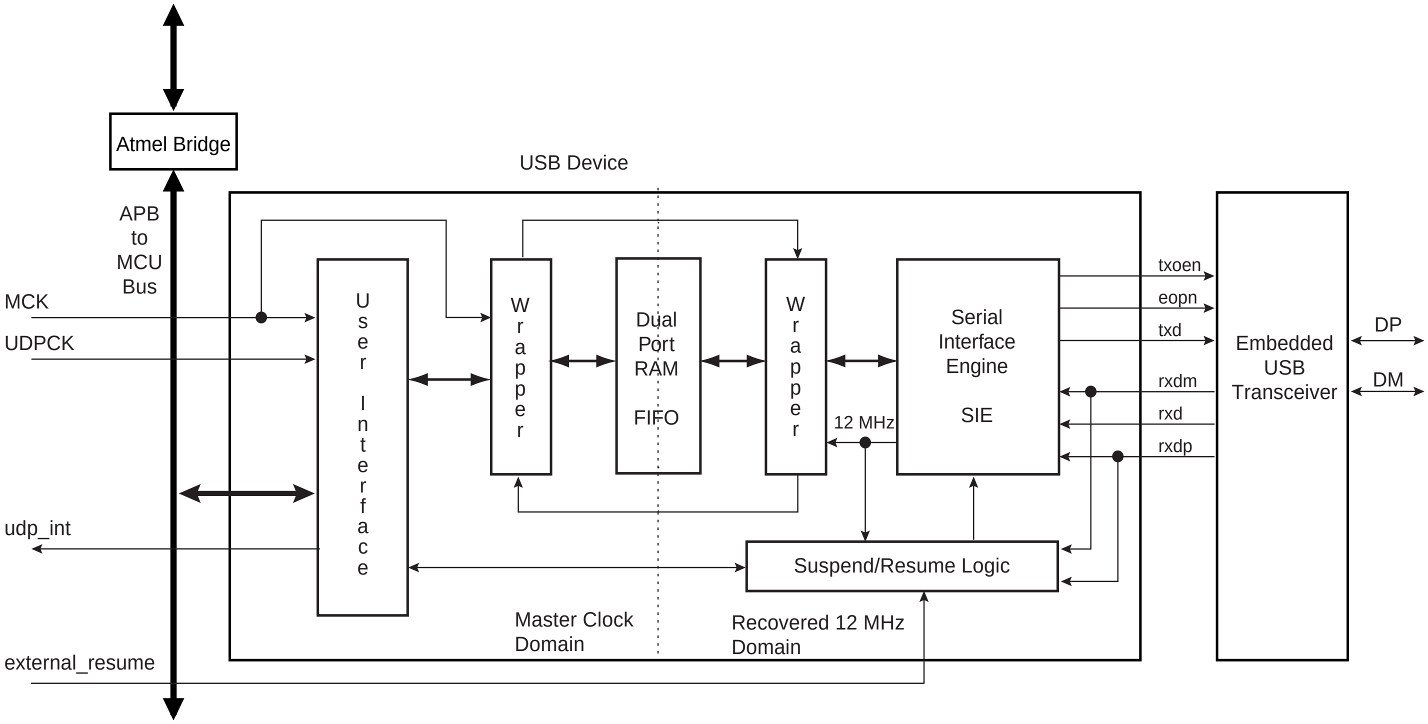


Figure 31-1. Block Diagram

Access to the UDP is via the APB bus interface. Read and write to the data FIFO are done by reading and writing 8-bit values to APB registers.

The UDP peripheral requires two clocks: one peripheral clock used by the MCK domain and a 48 MHz clock used by the 12 MHz domain.

A USB 2.0 full-speed pad is embedded and controlled by the Serial Interface Engine (SIE).

The signal external_resume is optional. It allows the UDP peripheral to wake up once in system mode. The host is then notified that the device asks for a resume. This optional feature must be also negotiated with the host during the enumeration.

31.3 Product Dependencies

For further details on the USB Device hardware implementation, see the specific Product Properties document.

The USB physical transceiver is integrated into the product. The bidirectional differential signals DP and DM are available from the product boundary.

One I/O line may be used by the application to check that VBUS is still available from the host. Self-powered devices may use this entry to be notified that the host has been powered off. In this case, the pullup on DP must be disabled in order to prevent feeding current to the host. The application should disconnect the transceiver, then remove the pullup.

31.3.1 I/O Lines

DP and DM are not controlled by any PIO controllers. The embedded USB physical transceiver is controlled by the USB device peripheral.

To reserve an I/O line to check VBUS, the programmer must first program the PIO controller to assign this I/O in input PIO mode.

31.3.2 Power Management

The USB device peripheral requires a 48 MHz clock. This clock must be generated by a PLL with an accuracy of ± 0.25%.

Thus, the USB device receives two clocks from the Power Management Controller (PMC): the master clock, MCK, used to drive the peripheral user interface, and the UDPCK, used to interface with the bus USB signals (recovered 12 MHz domain).

WARNING: The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP_TXCV register.

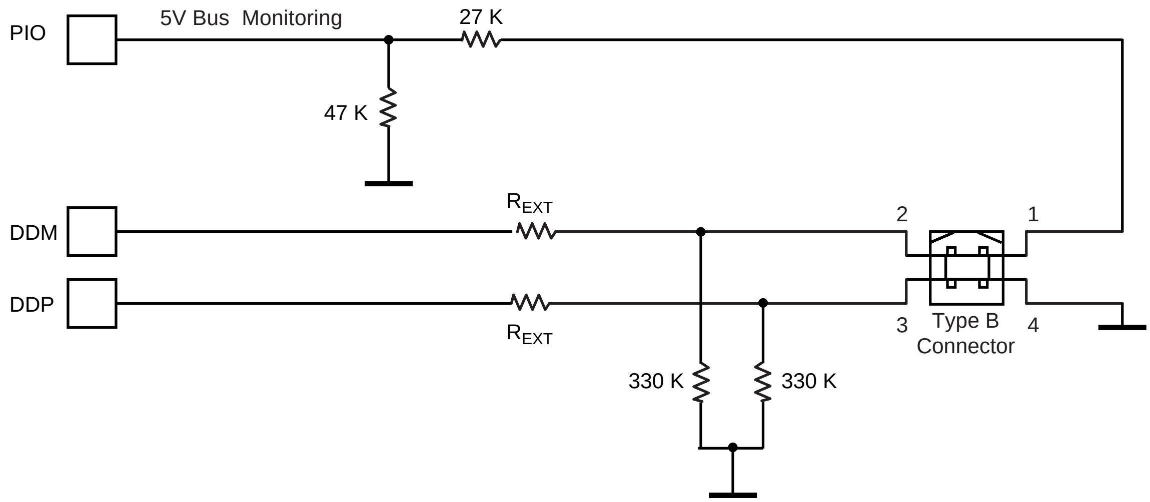
31.3.3 Interrupt

The USB device interface has an interrupt line connected to the Advanced Interrupt Controller (AIC).

Handling the USB device interrupt requires programming the AIC before configuring the UDP.

31.4 Typical Connection

Figure 31-2. Board Schematic to Interface Device Peripheral



31.4.1 USB Device Transceiver

The USB device transceiver is embedded in the product. A few discrete components are required as follows:

- the application detects all device states as defined in chapter 9 of the USB specification;
  - VBUS monitoring
- to reduce power consumption the host is disconnected
- for line termination.

31.4.2 VBUS Monitoring

VBUS monitoring is required to detect host connection. VBUS monitoring is done using a standard PIO with internal pullup disabled. When the host is switched off, it should be considered as a disconnect, the pullup must be disabled in order to prevent powering the host through the pull-up resistor.

When the host is disconnected and the transceiver is enabled, then DDP and DDM are floating. This may lead to over consumption. A solution is to connect 330 KΩ pulldowns on DP and DM. These pulldowns do not alter DDP and DDM signal integrity.

A termination serial resistor must be connected to DP and DM. The resistor value is defined in the electrical specification of the product (R_{EXT}).

31.5 Functional Description

31.5.1 USB V2.0 Full-speed Introduction

The USB V2.0 full-speed provides communication services between host and attached USB devices. Each device is offered with a collection of communication flows (pipes) associated with each endpoint. Software on the host communicates with a USB device through a set of communication flows.

Figure 31-3. Example of USB V2.0 Full-speed Communication Control

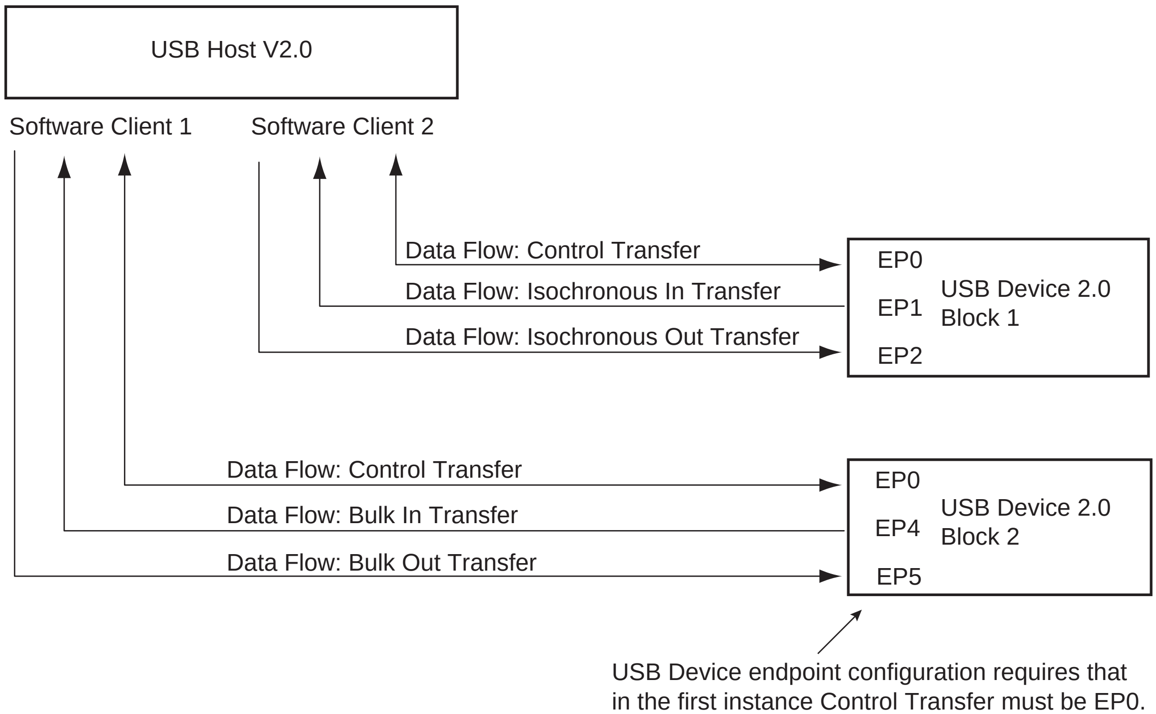
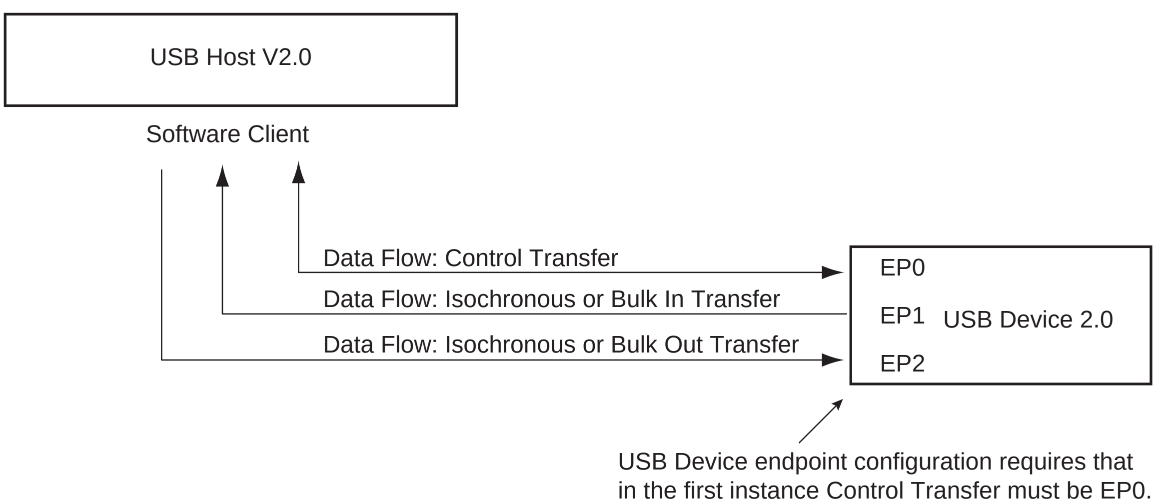


Figure 31-4. Example of USB V2.0 Full-speed Communication Control



The Control Transfer endpoint EP0 is always used when a USB device is first configured (USB v. 2.0 specifications).

31.5.1.1 USB V2.0 Full-speed Transfer Types

A communication flow is carried over one of four transfer types defined by the USB device.

Table 31-2. USB Communication Flow

Transfer	Direction	Bandwidth	Supported Endpoint Size	Error Detection	Retrying
Control	Bidirectional	Not guaranteed	8, 16, 32, 64	Yes	Automatic
Isochronous	Unidirectional	Guaranteed	1023	Yes	No
Interrupt	Unidirectional	Not guaranteed	≤ 64	Yes	Yes
Bulk	Unidirectional	Not guaranteed	8, 16, 32, 64	Yes	Yes

31.5.1.2 USB Bus Transactions

Each transfer results in one or more transactions over the USB bus. There are three kinds of transactions flowing across the bus in packets:

1. Setup Transaction
2. Data IN Transaction
3. Data OUT Transaction

31.5.1.3 USB Transfer Event Definitions

As indicated below, transfers are sequential events carried out on the USB bus.

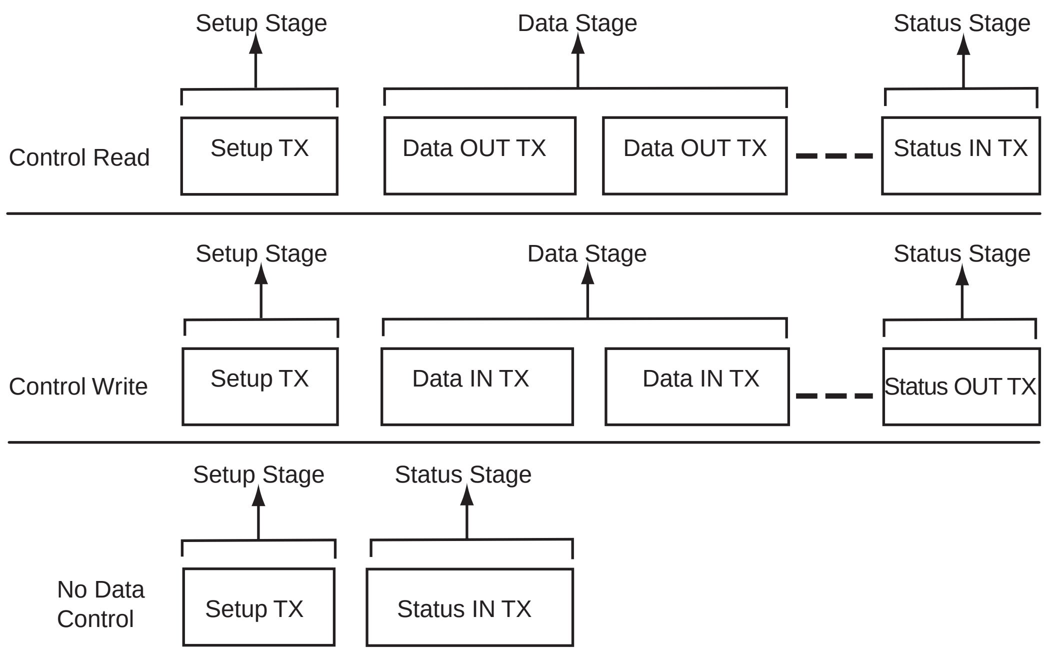
Table 31-3. USB Transfer Events

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.

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 31-5. Control Read and Write Sequences



Notes: 1. During the Status IN stage, the host waits for a zero length packet (Data IN transaction with no data) from the device using DATA1 PID. Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0,* for more information on the protocol layer.

2. During the Status OUT stage, the host emits a zero length packet to the device (Data OUT transaction with no data).

31.5.2 Handling Transactions with USB V2.0 Device Peripheral

31.5.2.1 Setup Transaction

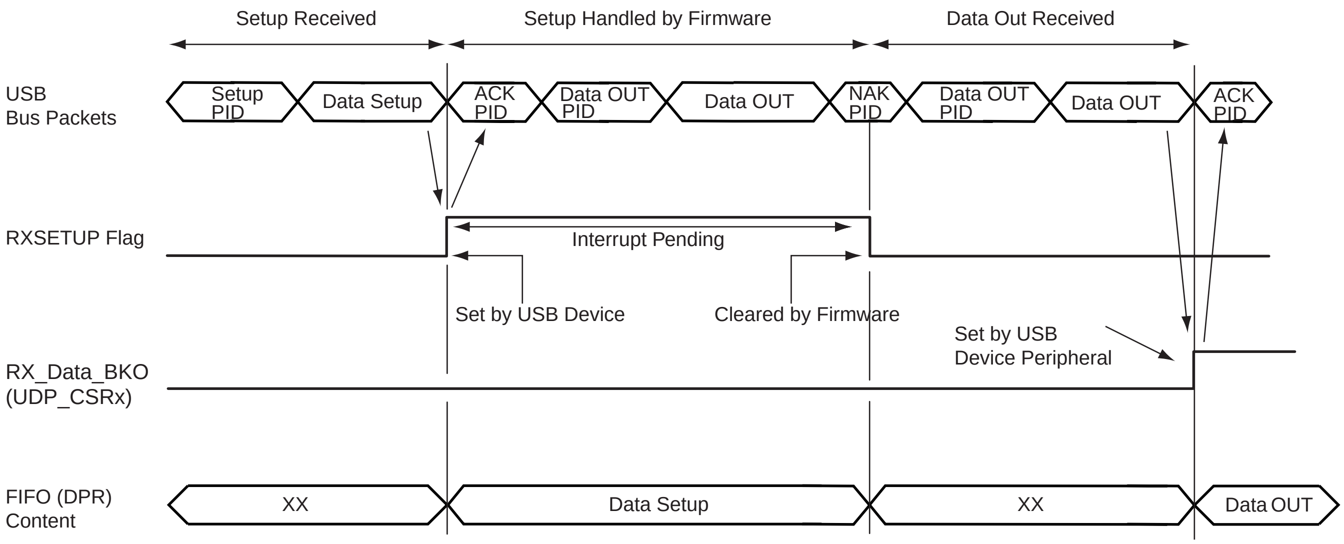
Setup is a special type of host-to-device transaction used during control transfers. Control transfers must be performed using endpoints with no ping-pong attributes. A setup transaction needs to be handled as soon as possible by the firmware. It is used to transmit requests from the host to the device. These requests are then handled by the USB device and may require more arguments. The arguments are sent to the device by a Data OUT transaction which follows the setup transaction. These requests may also return data. The data is carried out to the host by the next Data IN transaction which follows the setup transaction. A status transaction ends the control transfer.

When a setup transfer is received by the USB endpoint:

- The USB device automatically acknowledges the setup packet
- RXSETUP is set in the UDP_ CSRx register
- An endpoint interrupt is generated while the RXSETUP is not cleared. This interrupt is carried out to the microcontroller if interrupts are enabled for this endpoint.

Thus, firmware must detect the RXSETUP polling the UDP_ CSRx or catching an interrupt, read the setup packet in the FIFO, then clear the RXSETUP. RXSETUP cannot be cleared before the setup packet has been read in the FIFO. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the setup packet in the FIFO.

Figure 31-6. Setup Transaction Followed by a Data OUT Transaction



31.5.2.2 *Data IN Transaction*

Data IN transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the device to the host. Data IN transactions in isochronous transfer must be done using endpoints with ping-pong attributes.

31.5.2.2.1 *Using Endpoints Without Ping-pong Attributes*

To perform a Data IN transaction using a non ping-pong endpoint:

1. The application checks if it is possible to write in the FIFO by polling TXPKTRDY in the endpoint's UDP_ CSRx register (TXPKTRDY must be cleared).
2. The application writes the first packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP_ FDRx register,
3. The application notifies the USB peripheral it has finished by setting the TXPKTRDY in the endpoint's UDP_ CSRx register.
4. The application is notified that the endpoint's FIFO has been released by the USB device when TXCOMP in the endpoint's UDP_ CSRx register has been set. Then an interrupt for the corresponding endpoint is pending while TXCOMP is set.
5. The microcontroller writes the second packet of data to be sent in the endpoint's FIFO, writing zero or more byte values in the endpoint's UDP_ FDRx register,
6. The microcontroller notifies the USB peripheral it has finished by setting the TXPK-TRDY in the endpoint's UDP_ CSRx register.
7. The application clears the TXCOMP in the endpoint's UDP_ CSRx.

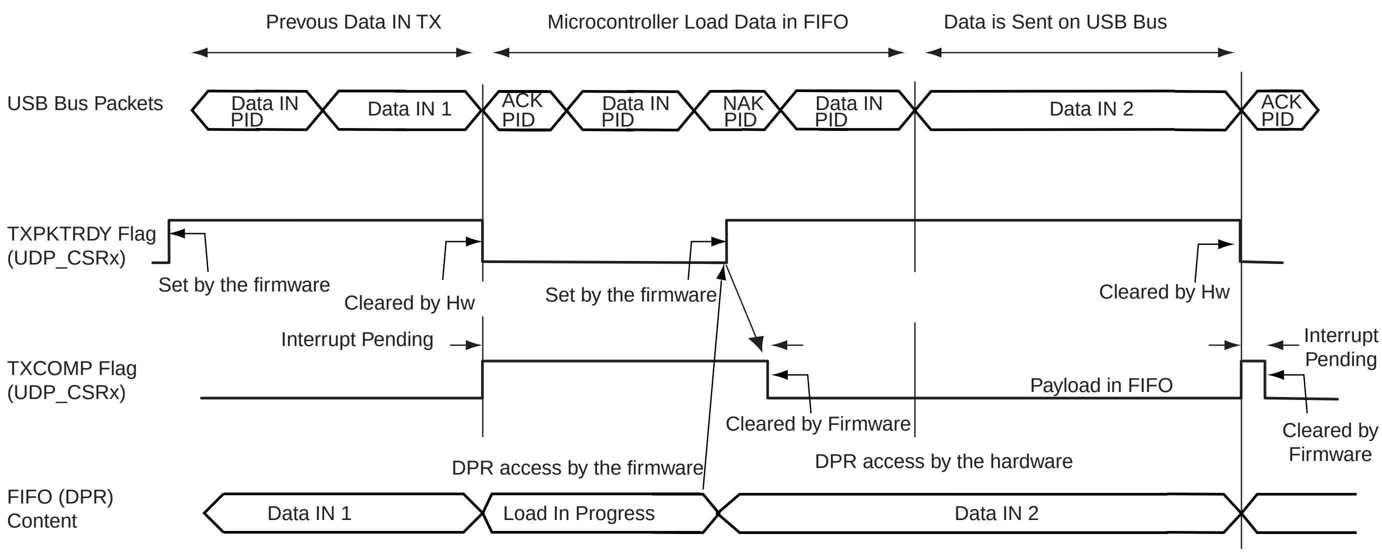
After the last packet has been sent, the application must clear TXCOMP once this has been set.

TXCOMP is set by the USB device when it has received an ACK PID signal for the Data IN packet. An interrupt is pending while TXCOMP is set.

Warning: TX_COMP must be cleared after TX_PKTRDY has been set.

Note: Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the Data IN protocol layer.

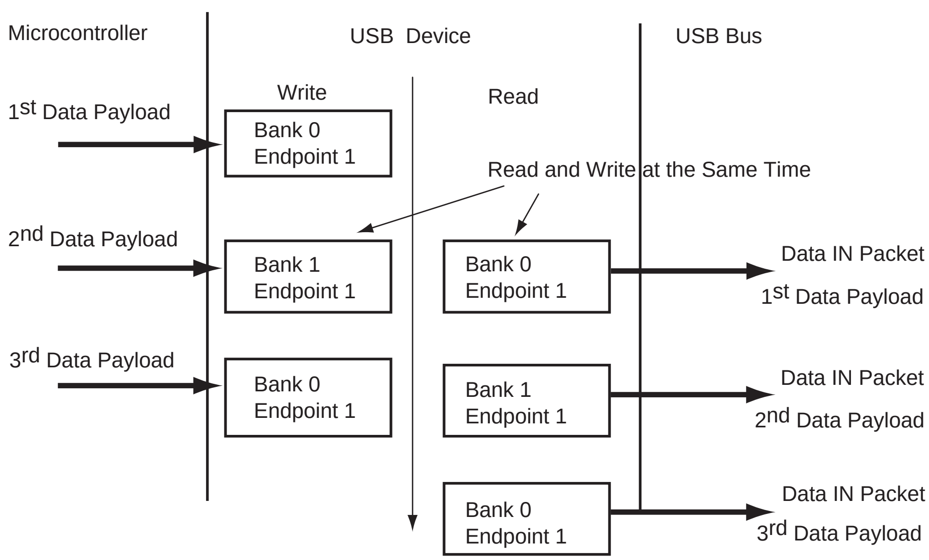
Figure 31-7. Data IN Transfer for Non Ping-pong Endpoint



31.5.2.2.1 *Using Endpoints With Ping-pong Attribute*

The use of an endpoint with ping-pong attributes is necessary during isochronous transfer. This also allows handling the maximum bandwidth defined in the USB specification during bulk transfer. To be able to guarantee a constant or the maximum bandwidth, the microcontroller must prepare the next data payload to be sent while the current one is being sent by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

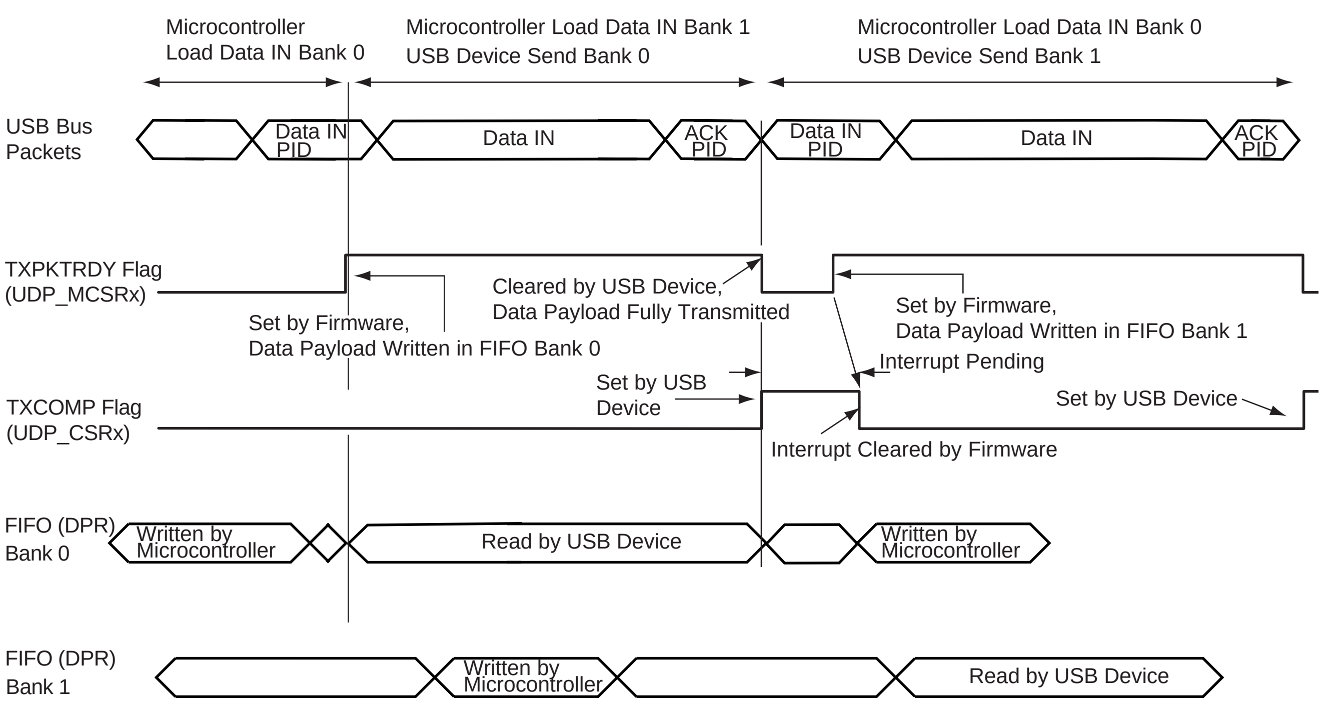
Figure 31-8. Bank Swapping Data IN Transfer for Ping-pong Endpoints



When using a ping-pong endpoint, the following procedures are required to perform Data IN transactions:

1. The microcontroller checks if it is possible to write in the FIFO by polling TXPKTRDY to be cleared in the endpoint's UDP_ CSRx register.
2. The microcontroller writes the first data payload to be sent in the FIFO (Bank 0), writing zero or more byte values in the endpoint's UDP_ FDRx register.
3. The microcontroller notifies the USB peripheral it has finished writing in Bank 0 of the FIFO by setting the TXPKTRDY in the endpoint's UDP_ CSRx register.
4. Without waiting for TXPKTRDY to be cleared, the microcontroller writes the second data payload to be sent in the FIFO (Bank 1), writing zero or more byte values in the endpoint's UDP_ FDRx register.
5. The microcontroller is notified that the first Bank has been released by the USB device when TXCOMP in the endpoint's UDP_ CSRx register is set. An interrupt is pending while TXCOMP is being set.
6. Once the microcontroller has received TXCOMP for the first Bank, it notifies the USB device that it has prepared the second Bank to be sent rising TXPKTRDY in the end-point's UDP_ CSRx register.
7. At this step, Bank 0 is available and the microcontroller can prepare a third data pay-load to be sent.

Figure 31-9. Data IN Transfer for Ping-pong Endpoint



Warning: There is software critical path due to the fact that once the second bank is filled, the driver has to wait for TX_COMP to set TX_PKTRDY. If the delay between receiving TX_COMP is set and TX_PKTRDY is set is too long, some Data IN packets may be NACKed, reducing the bandwidth.

Warning: TX_COMP must be cleared after TX_PKTRDY has been set.

31.5.2.3 Data OUT Transaction

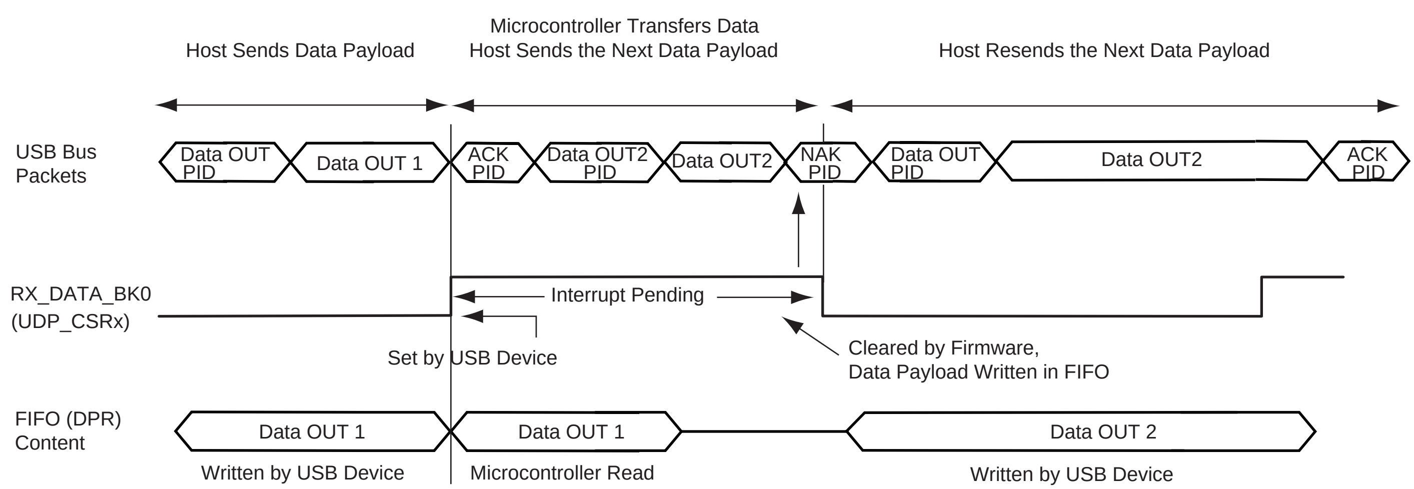
Data OUT transactions are used in control, isochronous, bulk and interrupt transfers and conduct the transfer of data from the host to the device. Data OUT transactions in isochronous transfers must be done using endpoints with ping-pong attributes.

31.5.2.3.1 Data OUT Transaction Without Ping-pong Attributes

To perform a Data OUT transaction, using a non ping-pong endpoint:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. While the FIFO associated to this endpoint is being used by the microcontroller, a NAK PID is returned to the host. Once the FIFO is available, data are written to the FIFO by the USB device and an ACK is automatically carried out to the host.
3. The microcontroller is notified that the USB device has received a data payload polling RX_DATA_BK0 in the endpoint's UDP_ CSRx register. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.
4. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP_ CSRx register.
5. The microcontroller carries out data received from the endpoint's memory to its memory. Data received is available by reading the endpoint's UDP_ FDRx register.
6. The microcontroller notifies the USB device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's UDP_ CSRx register.
7. A new Data OUT packet can be accepted by the USB device.

Figure 31-10. Data OUT Transfer for Non Ping-pong Endpoints

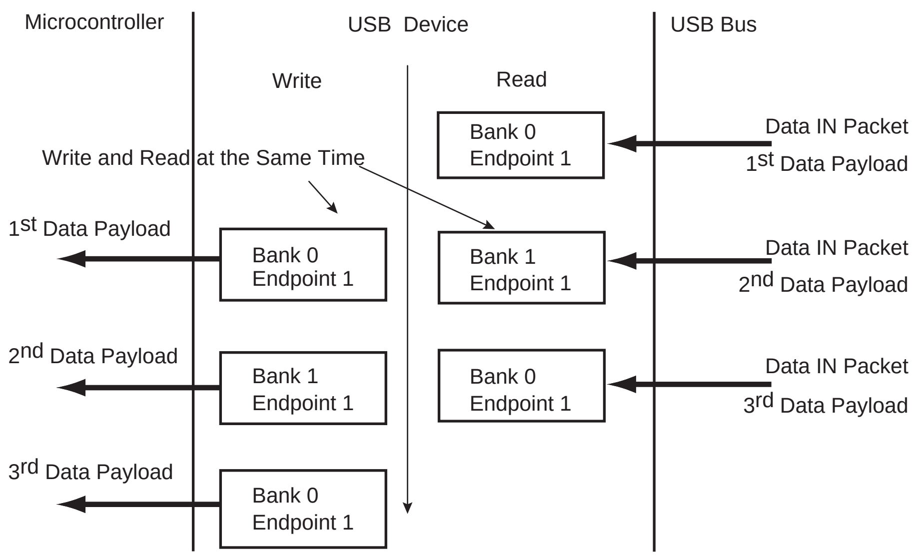


An interrupt is pending while the flag RX_DATA_BK0 is set. Memory transfer between the USB device, the FIFO and microcontroller memory can not be done after RX_DATA_BK0 has been cleared. Otherwise, the USB device would accept the next Data OUT transfer and overwrite the current Data OUT packet in the FIFO.

31.5.2.3.1 Using Endpoints With Ping-pong Attributes

During isochronous transfer, using an endpoint with ping-pong attributes is obligatory. To be able to guarantee a constant bandwidth, the microcontroller must read the previous data payload sent by the host, while the current data payload is received by the USB device. Thus two banks of memory are used. While one is available for the microcontroller, the other one is locked by the USB device.

Figure 31-11. Bank Swapping in Data OUT Transfers for Ping-pong Endpoints

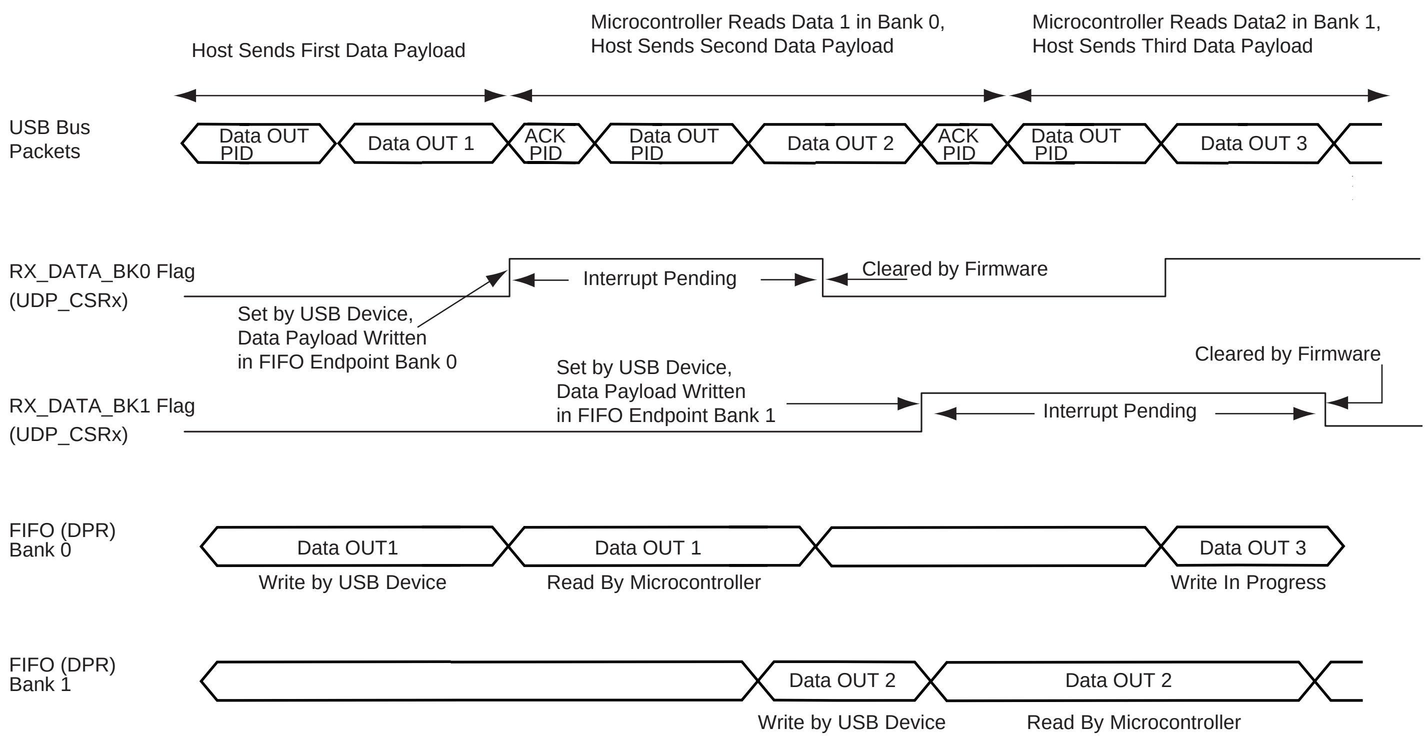


When using a ping-pong endpoint, the following procedures are required to perform Data OUT transactions:

1. The host generates a Data OUT packet.
2. This packet is received by the USB device endpoint. It is written in the endpoint's FIFO Bank 0.
3. The USB device sends an ACK PID packet to the host. The host can immediately send a second Data OUT packet. It is accepted by the device and copied to FIFO Bank 1.
4. The microcontroller is notified that the USB device has received a data payload, polling RX_DATA_BK0 in the endpoint's UDP_ CSRx register. An interrupt is pending for this endpoint while RX_DATA_BK0 is set.
5. The number of bytes available in the FIFO is made available by reading RXBYTECNT in the endpoint's UDP_ CSRx register.
6. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is made available by reading the endpoint's UDP_ FDRx register.
7. The microcontroller notifies the USB peripheral device that it has finished the transfer by clearing RX_DATA_BK0 in the endpoint's UDP_ CSRx register.
8. A third Data OUT packet can be accepted by the USB peripheral device and copied in the FIFO Bank 0.
9. If a second Data OUT packet has been received, the microcontroller is notified by the flag RX_DATA_BK1 set in the endpoint's UDP_ CSRx register. An interrupt is pending for this endpoint while RX_DATA_BK1 is set.

10. The microcontroller transfers out data received from the endpoint's memory to the microcontroller's memory. Data received is available by reading the endpoint's UDP_ FDRx register.
11. The microcontroller notifies the USB device it has finished the transfer by clearing RX_DATA_BK1 in the endpoint's UDP_ CSRx register.
12. A fourth Data OUT packet can be accepted by the USB device and copied in the FIFO Bank 0.

Figure 31-12. 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.

31.5.2.4 Stall Handshake

A stall handshake can be used in one of two distinct occasions. (For more information on the stall handshake, refer to Chapter 8 of the *Universal Serial Bus Specification, Rev 2.0.*)

- A functional stall is used when the halt feature associated with the endpoint is set. (Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0,* for more information on the halt feature.)
- To abort the current request, a protocol stall is used, but uniquely with control transfer.

The following procedure generates a stall packet:

1. The microcontroller sets the FORCESTALL flag in the UDP_ CSRx endpoint's register.
2. The host receives the stall packet.
3. The microcontroller is notified that the device has sent the stall by polling the STALLSENT to be set. An endpoint interrupt is pending while STALLSENT is set. The microcontroller must clear STALLSENT to clear the interrupt.

When a setup transaction is received after a stall handshake, STALLSENT must be cleared in order to prevent interrupts due to STALLSENT being set.

Figure 31-13. Stall Handshake (Data IN Transfer)

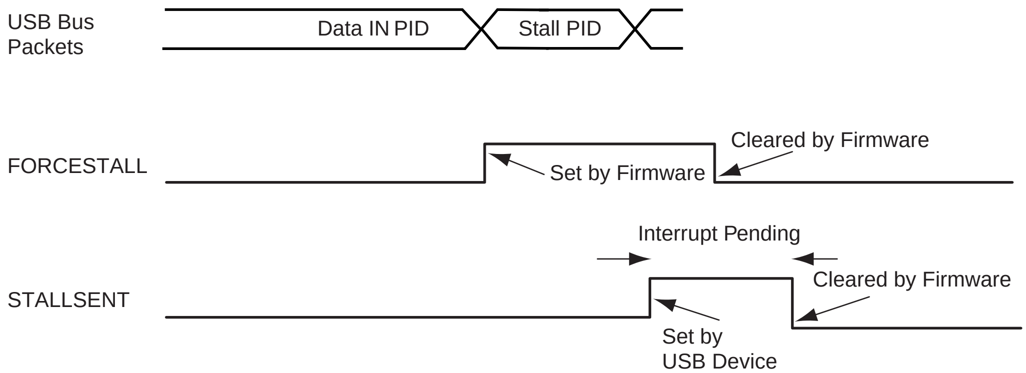
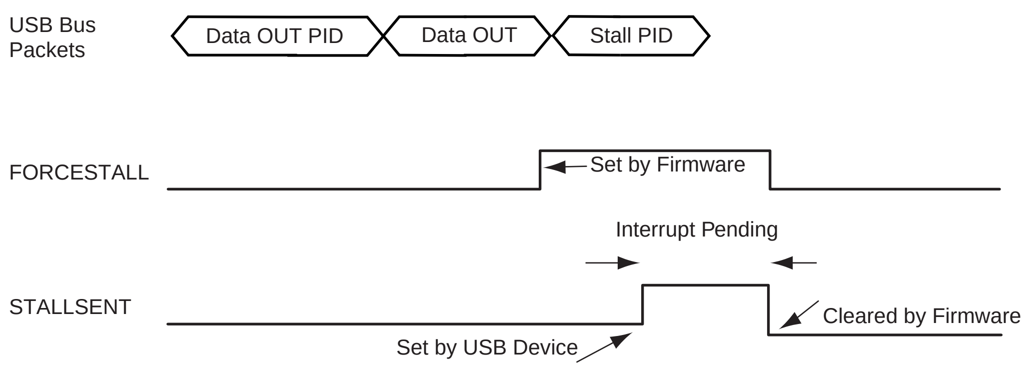


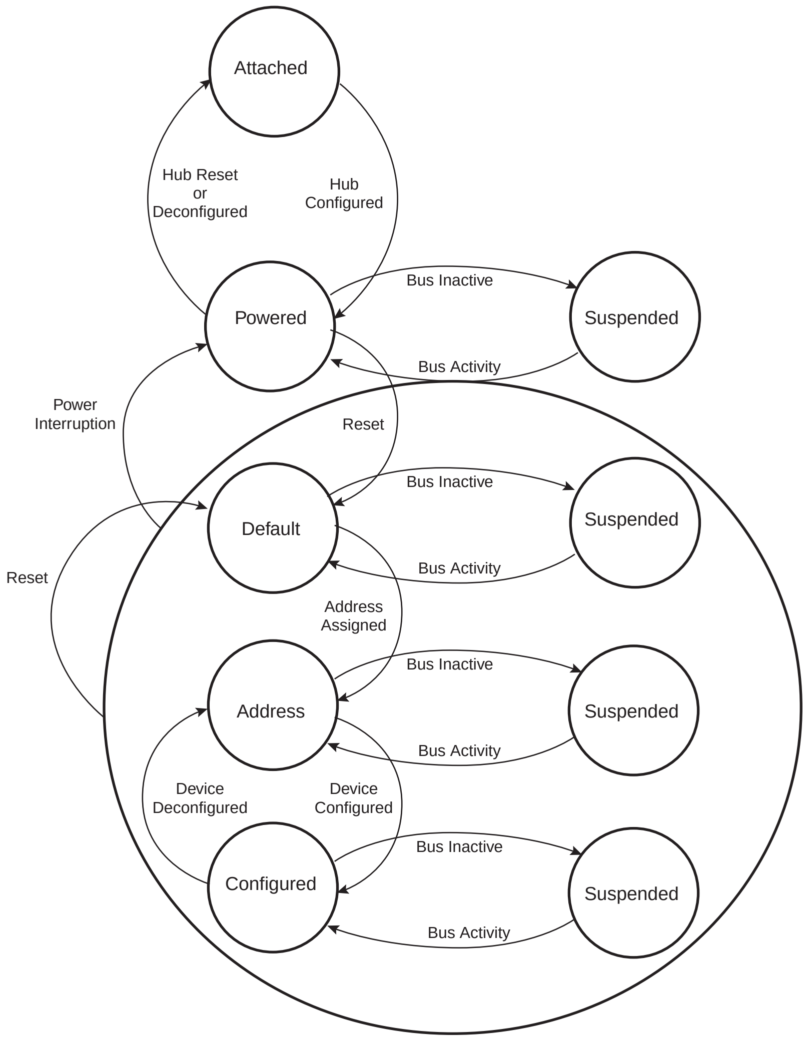
Figure 31-14. Stall Handshake (Data OUT Transfer)



31.5.3 Controlling Device States

A USB device has several possible states. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev 2.0*.

Figure 31-15. USB Device State Diagram



Movement from one state to another depends on the USB bus state or on standard requests sent through control transactions via the default endpoint (endpoint 0).

After a period of bus inactivity, the USB device enters Suspend Mode. Accepting Suspend/Resume requests from the USB host is mandatory. Constraints in Suspend Mode are very strict for bus-powered applications; devices may not consume more than 500 µA 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.

31.5.3.1 Not Powered State

Self powered devices can detect 5V VBUS using a PIO as described in the typical connection section. When the device is not connected to a host, device power consumption can be reduced by disabling MCK for the UDP, disabling UDPCK and disabling the transceiver. DDP and DDM lines are pulled down by 330 KΩ resistors.

31.5.3.2 Entering Attached State

When no device is connected, the USB DP and DM signals are tied to GND by 15 KΩ pull-down resistors integrated in the hub downstream ports. When a device is attached to a hub downstream port, the device connects a 1.5 KΩ pull-up resistor on DP. The USB bus line goes into IDLE state, DP is pulled up by the device 1.5 KΩ resistor to 3.3V and DM is pulled down by the 15 KΩ resistor of the host. To enable integrated pullup, the PUON bit in the UDP_TXVC register must be set.

Warning: To write to the UDP_TXVC register, MCK clock must be enabled on the UDP. This is done in the Power Management Controller.

After pullup connection, the device enters the powered state. In this state, the UDPCK and MCK must be enabled in the Power Management Controller. The transceiver can remain disabled.

31.5.3.3 From Powered State to Default State

After its connection to a USB host, the USB device waits for an end-of-bus reset. The unmaskable flag ENDBUSRES is set in the register UDP_ISR and an interrupt is triggered.

Once the ENDBUSRES interrupt has been triggered, the device enters Default State. In this state, the UDP software must:

- Enable the default endpoint, setting the EPEDS flag in the UDP_CSR[0] register and, optionally, enabling the interrupt for endpoint 0 by writing 1 to the UDP_IER register. The enumeration then begins by a control transfer.
- Configure the interrupt mask register which has been reset by the USB reset detection
- Enable the transceiver clearing the TXVDIS flag in the UDP_TXVC register.

In this state UDPCK and MCK must be enabled.

Warning: Each time an ENDBUSRES interrupt is triggered, the Interrupt Mask Register and UDP_CSR registers have been reset.

31.5.3.4 From Default State to Address State

After a set address standard device request, the USB host peripheral enters the address state.

Warning: Before the device enters in address state, it must achieve the Status IN transaction of the control transfer, i.e., the UDP device sets its new address once the TXCOMP flag in the UDP_CSR[0] register has been received and cleared.

To move to address state, the driver software sets the FADDEN flag in the UDP_GLB_STAT register, sets its new address, and sets the FEN bit in the UDP_FADDR register.

31.5.3.5 From Address State to Configured State

Once a valid Set Configuration standard request has been received and acknowledged, the device enables endpoints corresponding to the current configuration. This is done by setting the EPEDS and EPTYPE fields in the UDP_CSRx registers and, optionally, enabling corresponding interrupts in the UDP_IER register.

31.5.3.6 Entering in Suspend State

When a Suspend (no bus activity on the USB bus) is detected, the RXSUSP signal in the UDP_ISR register is set. This triggers an interrupt if the corresponding bit is set in the UDP_IMR register.This flag is cleared by writing to the UDP_ICR register. Then the device enters Suspend Mode.

In this state bus powered devices must drain less than 500uA from the 5V VBUS. As an example, the microcontroller switches to slow clock, disables the PLL and main oscillator, and goes into Idle Mode. It may also switch off other devices on the board.

The USB device peripheral clocks can be switched off. Resume event is asynchronously detected. MCK and UDPCK can be switched off in the Power Management controller and the USB transceiver can be disabled by setting the TXVDIS field in the UDP_TXVC register.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. Switching off MCK for the UDP peripheral must be one of the last operations after writing to the UDP_TXVC and acknowledging the RXSUSP.

31.5.3.7 Receiving a Host Resume

In suspend mode, a resume event on the USB bus line is detected asynchronously, transceiver and clocks are disabled (however the pullup shall not be removed).

Once the resume is detected on the bus, the WAKEUP signal in the UDP_ISR is set. It may generate an interrupt if the corresponding bit in the UDP_IMR register is set. This interrupt may be used to wake up the core, enable PLL and main oscillators and configure clocks.

Warning: Read, write operations to the UDP registers are allowed only if MCK is enabled for the UDP peripheral. MCK for the UDP must be enabled before clearing the WAKEUP bit in the UDP_ICR register and clearing TXVDIS in the UDP_TXVC register.

31.5.3.8 Sending a Device Remote Wakeup

In Suspend state it is possible to wake up the host sending an external resume.

- The device must wait at least 5 ms after being entered in suspend before sending an external resume.
- The device has 10 ms from the moment it starts to drain current and it forces a K state to resume the host.
- The device must force a K state from 1 to 15 ms to resume the host

Before sending a K state to the host, MCK, UDPCK and the transceiver must be enabled. Then to enable the remote wakeup feature, the RMWUPE bit in the UDP_GLB_STAT register must be enabled. To force the K state on the line, a transition of the ESR bit from 0 to 1 has to be done in the UDP_GLB_STAT register. This transition must be accomplished by first writing a 0 in the ESR bit and then writing a 1.

The K state is automatically generated and released according to the USB 2.0 specification.

31.6 USB Device Port (UDP) User Interface

WARNING: The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP_TXCV register.

Table 31-4. UDP Memory Map

Offset	Register	Name	Access	Reset State
0x000	Frame Number Register	UDP_ FRM_NUM	Read	0x0000_0000
0x004	Global State Register	UDP_ GLB_STAT	Read/Write	0x0000_0000
0x008	Function Address Register	UDP_ FADDR	Read/Write	0x0000_0100
0x00C	Reserved	–	–	–
0x010	Interrupt Enable Register	UDP_ IER	Write	
0x014	Interrupt Disable Register	UDP_ IDR	Write	
0x018	Interrupt Mask Register	UDP_ IMR	Read	0x0000_1200
0x01C	Interrupt Status Register	UDP_ ISR	Read	0x0000_XX00
0x020	Interrupt Clear Register	UDP_ ICR	Write	
0x024	Reserved	–	–	–
0x028	Reset Endpoint Register	UDP_ RST_EP	Read/Write	
0x02C	Reserved	–	–	–
0x030	Endpoint 0 Control and Status Register	UDP_CSR0	Read/Write	0x0000_0000
. . .	. . .			
See Note: (1)	Endpoint 7 Control and Status Register	UDP_CSR7	Read/Write	0x0000_0000
0x050	Endpoint 0 FIFO Data Register	UDP_ FDR0	Read/Write	0x0000_0000
. . .	. . .			
See Note: (2)	Endpoint 7 FIFO Data Register	UDP_ FDR7	Read/Write	0x0000_0000
0x070	Reserved	–	–	–
0x074	Transceiver Control Register	UDP_ TXVC (3)	Read/Write	0x0000_0100
0x078 - 0xFC	Reserved	–	–	–

Notes: 1. The addresses of the UDP_ CSRx registers are calculated as: 0x030 + 4(Endpoint Number - 1).
2. The addresses of the UDP_ FDRx registers are calculated as: 0x050 + 4(Endpoint Number - 1).
3. See Warning above the "UDP Memory Map" on this page.

31.6.1 UDP Frame Number Register

Register Name: UDP_ FRM_NUM

Access Type: Read-only

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

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

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

7	6	5	4	3	2	1	0
FRM_NUM							

- **FRM_NUM[10:0]: Frame Number as Defined in the Packet Field Formats**

This 11-bit value is incremented by the host on a per frame basis. This value is updated at each start of frame.

Value Updated at the SOF_EOP (Start of Frame End of Packet).

- **FRM_ERR: Frame Error**

This bit is set at SOF_EOP when the SOF packet is received containing an error.

This bit is reset upon receipt of SOF_PID.

- **FRM_OK: Frame OK**

This bit is set at SOF_EOP when the SOF packet is received without any error.

This bit is reset upon receipt of SOF_PID (Packet Identification).

In the Interrupt Status Register, the SOF interrupt is updated upon receiving SOF_PID. This bit is set without waiting for EOP.

Note: In the 8-bit Register Interface, FRM_OK is bit 4 of FRM_NUM_H and FRM_ERR is bit 3 of FRM_NUM_L.

31.6.2 UDP Global State Register

Register Name:UDP_ GLB_STAT

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	RMWUPE	RSMINPR	ESR	CONFG	FADDEN

This register is used to get and set the device state as specified in Chapter 9 of the *USB Serial Bus Specification, Rev.2.0*.

- **FADDEN: Function Address Enable**

Read:

0 = Device is not in address state.

1 = Device is in address state.

Write:

0 = No effect, only a reset can bring back a device to the default state.

1 = Sets device in address state. This occurs after a successful Set Address request. Beforehand, the UDP_ FADDR register must have been initialized with Set Address parameters. Set Address must complete the Status Stage before setting FADDEN. Refer to chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **CONFG: Configured**

Read:

0 = Device is not in configured state.

1 = Device is in configured state.

Write:

0 = Sets device in a non configured state

1 = Sets device in configured state.

The device is set in configured state when it is in address state and receives a successful Set Configuration request. Refer to Chapter 9 of the *Universal Serial Bus Specification, Rev. 2.0* for more details.

- **ESR: Enable Send Resume**

0 = Mandatory value prior to starting any Remote Wake Up procedure.

1 = Starts the Remote Wake Up procedure if this bit value was 0 and if RMWUPE is enabled.

- **RMWUPE: Remote Wake Up Enable**

0 = The Remote Wake Up feature of the device is disabled.

1 = The Remote Wake Up feature of the device is enabled.

31.6.3 UDP Function Address Register

Register Name: UDP_ FADDR

Access Type: Read/Write

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

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

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

7	6	5	4	3	2	1	0
–	FADD						

- **FADD[6:0]: Function Address Value**

The Function Address Value must be programmed by firmware once the device receives a set address request from the host, and has achieved the status stage of the no-data control sequence. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information. After power up or reset, the function address value is set to 0.

- **FEN: Function Enable**

Read:

0 = Function endpoint disabled.

1 = Function endpoint enabled.

Write:

0 = Disables function endpoint.

1 = Default value.

The Function Enable bit (FEN) allows the microcontroller to enable or disable the function endpoints. The microcontroller sets this bit after receipt of a reset from the host. Once this bit is set, the USB device is able to accept and transfer data packets from and to the host.

31.6.4 UDP Interrupt Enable Register

Register Name:UDP_IER

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	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 = Enables corresponding Endpoint Interrupt.

- **RXSUSP: Enable UDP Suspend Interrupt**

0 = No effect.

1 = Enables UDP Suspend Interrupt.

- **RXRSM: Enable UDP Resume Interrupt**

0 = No effect.

1 = Enables UDP Resume Interrupt.

- **SOFINT: Enable Start Of Frame Interrupt**

0 = No effect.

1 = Enables Start Of Frame Interrupt.

- **WAKEUP: Enable UDP bus Wakeup Interrupt**

0 = No effect.

1 = Enables USB bus Interrupt.

31.6.5 UDP Interrupt Disable Register

Register Name: UDP_IDR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	–	SOFINT	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 = Disables corresponding Endpoint Interrupt.

- **RXSUSP: Disable UDP Suspend Interrupt**

0 = No effect.

1 = Disables UDP Suspend Interrupt.

- **RXRSM: Disable UDP Resume Interrupt**

0 = No effect.

1 = Disables UDP Resume Interrupt.

- **SOFINT: Disable Start Of Frame Interrupt**

0 = No effect.

1 = Disables Start Of Frame Interrupt

- **WAKEUP: Disable USB Bus Interrupt**

0 = No effect.

1 = Disables USB Bus Wakeup Interrupt.

31.6.6 UDP Interrupt Mask Register

Register Name: UDP_IMR

Access Type: Read-only

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

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

15	14	13	12[1]	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

Note: 1. Bit 12 of UDP_IMR cannot be masked and is always read at 1.

- **EP0INT: Mask Endpoint 0 Interrupt**
- **EP1INT: Mask Endpoint 1 Interrupt**
- **EP2INT: Mask Endpoint 2 Interrupt**
- **EP3INT: Mask Endpoint 3 Interrupt**
- **EP4INT: Mask Endpoint 4 Interrupt**
- **EP5INT: Mask Endpoint 5 Interrupt**
- **EP6INT: Mask Endpoint 6 Interrupt**
- **EP7INT: Mask Endpoint 7 Interrupt**

0 = Corresponding Endpoint Interrupt is disabled.

1 = Corresponding Endpoint Interrupt is enabled.

- **RXSUSP: Mask UDP Suspend Interrupt**

0 = UDP Suspend Interrupt is disabled.

1 = UDP Suspend Interrupt is enabled.

- **RXRSM: Mask UDP Resume Interrupt.**

0 = UDP Resume Interrupt is disabled.

1 = UDP Resume Interrupt is enabled.

- **SOFINT: Mask Start Of Frame Interrupt**

0 = Start of Frame Interrupt is disabled.

1 = Start of Frame Interrupt is enabled.

- **WAKEUP: USB Bus WAKEUP Interrupt**

0 = USB Bus Wakeup Interrupt is disabled.

1 = USB Bus Wakeup Interrupt is enabled.

Note: When the USB block is in suspend mode, the application may power down the USB logic. In this case, any USB HOST resume request that is made must be taken into account and, thus, the reset value of the RXRSM bit of the register UDP_ IMR is enabled.

31.6.7 UDP Interrupt Status Register

Register Name: UDP_ ISR

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
–	–	WAKEUP	ENDBUSRES	SOFINT	EXTRSM	RXRSM	RXSUSP

7	6	5	4	3	2	1	0
EP7INT	EP6INT	EP5INT	EP4INT	EP3INT	EP2INT	EP1INT	EP0INT

- **EP0INT: Endpoint 0 Interrupt Status**
- **EP1INT: Endpoint 1 Interrupt Status**
- **EP2INT: Endpoint 2 Interrupt Status**
- **EP3INT: Endpoint 3 Interrupt Status**
- **EP4INT: Endpoint 4 Interrupt Status**
- **EP5INT: Endpoint 5 Interrupt Status**
- **EP6INT: Endpoint 6 Interrupt Status**
- **EP7INT: Endpoint 7Interrupt Status**

0 = No Endpoint0 Interrupt pending.

1 = Endpoint0 Interrupt has been raised.

Several signals can generate this interrupt. The reason can be found by reading UDP_ CSR0:

RXSETUP set to 1

RX_DATA_BK0 set to 1

RX_DATA_BK1 set to 1

TXCOMP set to 1

STALLSENT set to 1

EP0INT is a sticky bit. Interrupt remains valid until EP0INT is cleared by writing in the corresponding UDP_ CSR0 bit.

- **RXSUSP: UDP Suspend Interrupt Status**

0 = No UDP Suspend Interrupt pending.

1 = UDP Suspend Interrupt has been raised.

The USB device sets this bit when it detects no activity for 3ms. The USB device enters Suspend mode.

- **RXRSM: UDP Resume Interrupt Status**

0 = No UDP Resume Interrupt pending.

1 =UDP Resume Interrupt has been raised.

The USB device sets this bit when a UDP resume signal is detected at its port.

After reset, the state of this bit is undefined, the application must clear this bit by setting the RXRSM flag in the UDP_ ICR register.

- **SOFINT: Start of Frame Interrupt Status**

0 = No Start of Frame Interrupt pending.

1 = Start of Frame Interrupt has been raised.

This interrupt is raised each time a SOF token has been detected. It can be used as a synchronization signal by using

isochronous endpoints.

- **ENDBUSRES: End of BUS Reset Interrupt Status**

0 = No End of Bus Reset Interrupt pending.

1 = End of Bus Reset Interrupt has been raised.

This interrupt is raised at the end of a UDP reset sequence. The USB device must prepare to receive requests on the endpoint 0. The host starts the enumeration, then performs the configuration.

- **WAKEUP: UDP Resume Interrupt Status**

0 = No Wakeup Interrupt pending.

1 = A Wakeup Interrupt (USB Host Sent a RESUME or RESET) occurred since the last clear.

After reset the state of this bit is undefined, the application must clear this bit by setting the WAKEUP flag in the UDP_ ICR register.

31.6.8 UDP Interrupt Clear Register

Register Name: UDP_ ICR

Access Type: Write-only

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

- **RXSUSP: Clear UDP Suspend Interrupt**

0 = No effect.

1 = Clears UDP Suspend Interrupt.

- **RXRSM: Clear UDP Resume Interrupt**

0 = No effect.

1 = Clears UDP Resume Interrupt.

- **SOFINT: Clear Start Of Frame Interrupt**

0 = No effect.

1 = Clears Start Of Frame Interrupt.

- **ENDBUSRES: Clear End of Bus Reset Interrupt**

0 = No effect.

1 = Clears End of Bus Reset Interrupt.

- **WAKEUP: Clear Wakeup Interrupt**

0 = No effect.

1 = Clears Wakeup Interrupt.

31.6.9 UDP Reset Endpoint Register

Register Name: UDP_ RST_EP

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
EP7INT	EP6INT	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 UDP_ CSRx flags.

0 = No reset.

1 = Forces the corresponding endpoint FIF0 pointers to 0, therefore RXBYTECNT field is read at 0 in UDP_ CSRx register.

31.6.10 UDP Endpoint Control and Status Register

Register Name: UDP_ 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

WARNING: Due to synchronization between MCK and UDPCK, the software application must wait for the end of the write operation before executing another write by polling the bits which must be set/cleared.

```
//! Clear flags of UDP UDP_CSR register and waits for synchronization
#define Udp_ep_clr_flag(pInterface, endpoint, flags) { \
  while (pInterface->UDP_CSR[endpoint] & (flags)) \
    pInterface->UDP_CSR[endpoint] &= ~(flags); \
  }
//! Set flags of UDP UDP_CSR register and waits for synchronization
#define Udp_ep_set_flag(pInterface, endpoint, flags) { \
  while ( (pInterface->UDP_CSR[endpoint] & (flags)) != (flags) ) \
    pInterface->UDP_CSR[endpoint] |= (flags); \
  }
```

- **TXCOMP: Generates an IN Packet with Data Previously Written in the DPR**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Clear the flag, clear the interrupt.

1 = No effect.

Read (Set by the USB peripheral):

0 = Data IN transaction has not been acknowledged by the Host.

1 = Data IN transaction is achieved, acknowledged by the Host.

After having issued a Data IN transaction setting TXPKTRDY, the device firmware waits for TXCOMP to be sure that the host has acknowledged the transaction.

- **RX_DATA_BK0: Receive Data Bank 0**

This flag generates an interrupt while it is set to one.
Write (Cleared by the firmware):

0 = Notify USB peripheral device that data have been read in the FIFO's Bank 0.

1 = To leave the read value unchanged.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 0.

1 = A data packet has been received, it has been stored in the FIFO's Bank 0.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to the microcontroller memory. The number of bytes received is available in RXBYTCENT field. Bank 0 FIFO values are read through the UDP_ FDRx register. Once a transfer is done, the device firmware must release Bank 0 to the USB peripheral device by clearing RX_DATA_BK0.

- **RXSETUP: Received Setup**

This flag generates an interrupt while it is set to one.

Read:

0 = No setup packet available.

1 = A setup data packet has been sent by the host and is available in the FIFO.

Write:

0 = Device firmware notifies the USB peripheral device that it has read the setup data in the FIFO.

1 = No effect.

This flag is used to notify the USB device firmware that a valid Setup data packet has been sent by the host and successfully received by the USB device. The USB device firmware may transfer Setup data from the FIFO by reading the UDP_ FDRx register to the microcontroller memory. Once a transfer has been done, RXSETUP must be cleared by the device firmware.

Ensuing Data OUT transaction is not accepted while RXSETUP is set.

- **STALLSENT: Stall Sent (Control, Bulk Interrupt Endpoints)/ISOERROR (Isochronous Endpoints)**

This flag generates an interrupt while it is set to one.

STALLSENT: This ends a STALL handshake.

Read:

0 = The host has not acknowledged a STALL.

1 = Host has acknowledged the stall.

Write:

0 = Resets the STALLSENT flag, clears the interrupt.

1 = No effect.

This is mandatory for the device firmware to clear this flag. Otherwise the interrupt remains.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

ISOERROR: A CRC error has been detected in an isochronous transfer.

Read:

0 = No error in the previous isochronous transfer.

1 = CRC error has been detected, data available in the FIFO are corrupted.

Write:

0 = Resets the ISOERROR flag, clears the interrupt.

1 = No effect.

- **TXPKTRDY: Transmit Packet Ready**

This flag is cleared by the USB device.

This flag is set by the USB device firmware.

Read:

0 = Can be set to one to send the FIFO data.

1 = The data is waiting to be sent upon reception of token IN.

Write:

0 = Can be written if old value is zero.

1 = A new data payload is has been written in the FIFO by the firmware and is ready to be sent.

This flag is used to generate a Data IN transaction (device to host). Device firmware checks that it can write a data payload in the FIFO, checking that TXPKTRDY is cleared. Transfer to the FIFO is done by writing in the UDP_ FDRx register. Once the data payload has been transferred to the FIFO, the firmware notifies the USB device setting TXPKTRDY to one. USB bus transactions can start. TXCOMP is set once the data payload has been received by the host.

- **FORCESTALL: Force Stall (used by Control, Bulk and Isochronous Endpoints)**

Read:

0 = Normal state.

1 = Stall state.

Write:

0 = Return to normal state.

1 = Send STALL to the host.

Refer to chapters 8.4.5 and 9.4.5 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the STALL handshake.

Control endpoints: During the data stage and status stage, this bit indicates that the microcontroller cannot complete the request.

Bulk and interrupt endpoints: This bit notifies the host that the endpoint is halted.

The host acknowledges the STALL, device firmware is notified by the STALLSENT flag.

- **RX_DATA_BK1: Receive Data Bank 1 (only used by endpoints with ping-pong attributes)**

This flag generates an interrupt while it is set to one.

Write (Cleared by the firmware):

0 = Notifies USB device that data have been read in the FIFO’s Bank 1.

1 = To leave the read value unchanged.

Read (Set by the USB peripheral):

0 = No data packet has been received in the FIFO's Bank 1.

1 = A data packet has been received, it has been stored in FIFO's Bank 1.

When the device firmware has polled this bit or has been interrupted by this signal, it must transfer data from the FIFO to microcontroller memory. The number of bytes received is available in RXBYTECNT field. Bank 1 FIFO values are read through UDP_ FDRx register. Once a transfer is done, the device firmware must release Bank 1 to the USB device by clearing RX_DATA_BK1.

- **DIR: Transfer Direction (only available for control endpoints)**

Read/Write

0 = Allows Data OUT transactions in the control data stage.

1 = Enables Data IN transactions in the control data stage.

Refer to Chapter 8.5.3 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on the control data stage.

This bit must be set before UDP_ CSRx/RXSETUP is cleared at the end of the setup stage. According to the request sent in the setup data packet, the data stage is either a device to host (DIR = 1) or host to device (DIR = 0) data transfer. It is not necessary to check this bit to reverse direction for the status stage.

- **EPTYPE[2:0]: Endpoint Type**

Read/Write

000	Control
001	Isochronous OUT
101	Isochronous IN
010	Bulk OUT
110	Bulk IN
011	Interrupt OUT
111	Interrupt IN

- **DTGLE: Data Toggle**

Read-only

0 = Identifies DATA0 packet.

1 = Identifies DATA1 packet.

Refer to Chapter 8 of the *Universal Serial Bus Specification, Rev. 2.0* for more information on DATA0, DATA1 packet definitions.

- **EPEDS: Endpoint Enable Disable**

Read:

0 = Endpoint disabled.

1 = Endpoint enabled.

Write:

0 = Disables endpoint.

1 = Enables endpoint.

Control endpoints are always enabled. Reading or writing this field has no effect on control endpoints.

Note: After reset all endpoints are configured as control endpoints (zero).

- **RXBYTECNT[10:0]: Number of Bytes Available in the FIFO**

Read-only

When the host sends a data packet to the device, the USB device stores the data in the FIFO and notifies the microcontroller. The microcontroller can load the data from the FIFO by reading RXBYTECENT bytes in the UDP_ FDRx register.

31.6.11 UDP FIFO Data Register

Register Name:UDP_ 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 UDP_ CSRx register is the number of bytes to be read from the FIFO (sent by the host).

The maximum number of bytes to write is fixed by the Max Packet Size in the Standard Endpoint Descriptor. It can not be more than the physical memory size associated to the endpoint. Refer to the *Universal Serial Bus Specification, Rev. 2.0* for more information.

31.6.12 UDP Transceiver Control Register

Register Name:UDP_ TXVC

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
–	–	–	–	–	–	PUON	TXVDIS

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

WARNING: The UDP peripheral clock in the Power Management Controller (PMC) must be enabled before any read/write operations to the UDP registers including the UDP_TXCV register.

- **TXVDIS: Transceiver Disable**

When UDP is disabled, power consumption can be reduced significantly by disabling the embedded transceiver. This can be done by setting TXVDIS field.

To enable the transceiver, TXVDIS must be cleared.

- **PUON: Pullup On**

0: The 1.5KΩ integrated pullup on DP is disconnected.

1: The 1.5 KΩ integrated pullup on DP is connected.

NOTE: If the USB pullup is not connected on DP, the user should not write in any UDP register other than the UDP_ TXVC register. This is because if DP and DM are floating at 0, or pulled down, then SE0 is received by the device with the consequence of a USB Reset.

32. Ethernet MAC 10/100 (EMACB)

32.1 Description

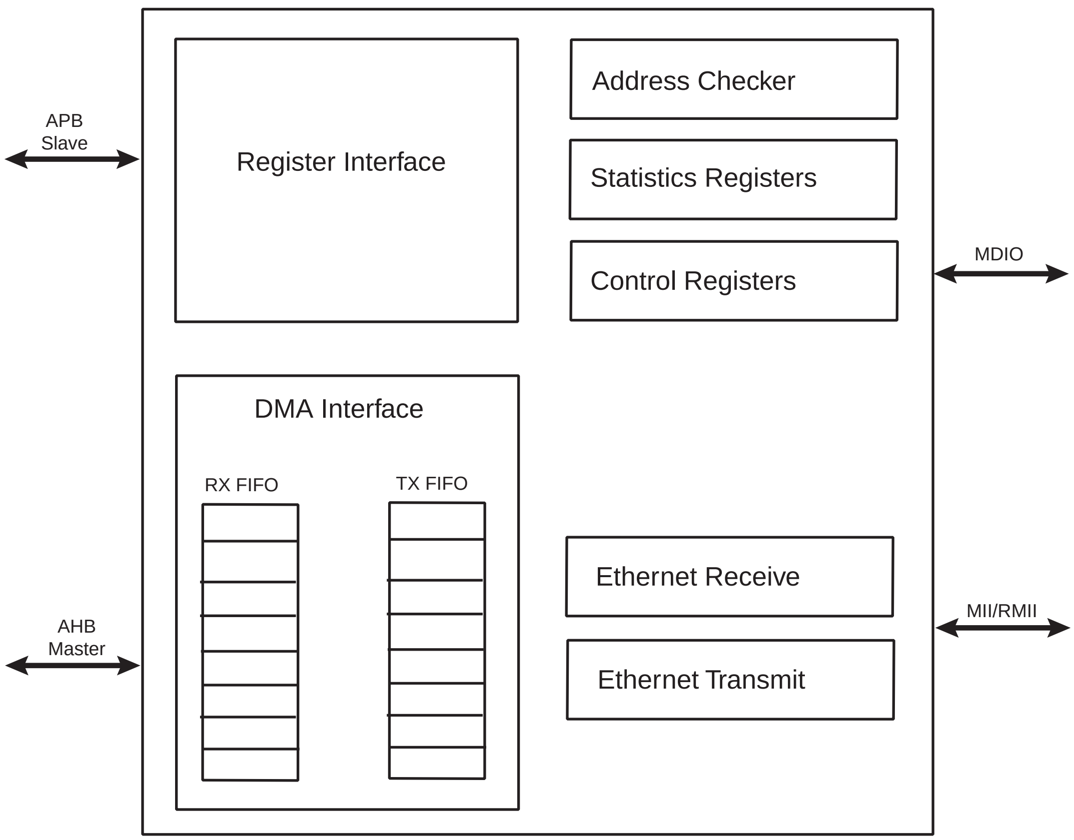
The EMAC module implements a 10/100 Ethernet MAC compatible with the IEEE 802.3 standard using an address checker, statistics and control registers, receive and transmit blocks, and a DMA interface.

The address checker recognizes four specific 48-bit addresses and contains a 64-bit hash register for matching multicast and unicast addresses. It can recognize the broadcast address of all ones, copy all frames, and act on an external address match signal.

The statistics register block contains registers for counting various types of event associated with transmit and receive operations. These registers, along with the status words stored in the receive buffer list, enable software to generate network management statistics compatible with IEEE 802.3.

32.2 Block Diagram

Figure 32-1. EMAC Block Diagram



32.3 Functional Description

The MACB has several clock domains:

- System bus clock (AHB and APB): DMA and register blocks
- Transmit clock: transmit block
- Receive clock: receive and address checker blocks

The only system constraint is 160 MHz for the system bus clock, above which MDC would toggle at above 2.5 MHz.

The system bus clock must run at least as fast as the receive clock and transmit clock (25 MHz at 100 Mbps, and 2.5 MHZ at 10 Mbps).

Figure 32-1 illustrates the different blocks of the EMAC module.

The control registers drive the MDIO interface, setup up DMA activity, start frame transmission and select modes of operation such as full- or half-duplex.

The receive block checks for valid preamble, FCS, alignment and length, and presents received frames to the address checking block and DMA interface.

The transmit block takes data from the DMA interface, adds preamble and, if necessary, pad and FCS, and transmits data according to the CSMA/CD (carrier sense multiple access with collision detect) protocol. The start of transmission is deferred if CRS (carrier sense) is active.

If COL (collision) becomes active during transmission, a jam sequence is asserted and the transmission is retried after a random back off. CRS and COL have no effect in full duplex mode.

The DMA block connects to external memory through its AHB bus interface. It contains receive and transmit FIFOs for buffering frame data. It loads the transmit FIFO and empties the receive FIFO using AHB bus master operations. Receive data is not sent to memory until the address checking logic has determined that the frame should be copied. Receive or transmit frames are stored in one or more buffers. Receive buffers have a fixed length of 128 bytes. Transmit buffers range in length between 0 and 2047 bytes, and up to 128 buffers are permitted per frame. The DMA block manages the transmit and receive framebuffer queues. These queues can hold multiple frames.

32.3.1 Memory Interface

Frame data is transferred to and from the EMAC through the DMA interface. All transfers are 32-bit words and may be single accesses or bursts of 2, 3 or 4 words. Burst accesses do not cross sixteen-byte boundaries. Bursts of 4 words are the default data transfer; single accesses or bursts of less than four words may be used to transfer data at the beginning or the end of a buffer.

The DMA controller performs six types of operation on the bus. In order of priority, these are:

1. Receive buffer manager write
2. Receive buffer manager read
3. Transmit data DMA read
4. Receive data DMA write
5. Transmit buffer manager read
6. Transmit buffer manager write

32.3.1.1 FIFO

The FIFO depths are **28** bytes and **28** bytes and area function of the system clock speed, memory latency and network speed.

Data is typically transferred into and out of the FIFOs in bursts of four words. For receive, a bus request is asserted when the FIFO contains four words and has space for three more. For transmit, a bus request is generated when there is space for four words, or when there is space for two words if the next transfer is to be only one or two words.

Thus the bus latency must be less than the time it takes to load the FIFO and transmit or receive three words (12 bytes) of data.

At 100 Mbit/s, it takes 960 ns to transmit or receive 12 bytes of data. In addition, six master clock cycles should be allowed for data to be loaded from the bus and to propagate through the FIFOs. For a 60 MHz master clock this takes 100 ns, making the bus latency requirement 860 ns.

32.3.1.2 Receive Buffers

Received frames, including CRC/FCS optionally, are written to receive buffers stored in memory. Each receive buffer is 128 bytes long. The start location for each receive buffer is stored in memory in a list of receive buffer descriptors at a location pointed to by the receive buffer queue pointer register. The receive buffer start location is a word address. For the first buffer of a frame, the start location can be offset by up to three bytes depending on the value written to bits 14 and 15 of the network configuration register. If the start location of the buffer is offset the available length of the first buffer of a frame is reduced by the corresponding number of bytes.

Each list entry consists of two words, the first being the address of the receive buffer and the second being the receive status. If the length of a receive frame exceeds the buffer length, the status word for the used buffer is written with zeroes except for the "start of frame" bit and the offset bits, if appropriate. Bit zero of the address field is written to one to show the buffer has been used. The receive buffer manager then reads the location of the next receive buffer and fills that with receive frame data. The final buffer descriptor status word contains the complete frame status. Refer to Table 32-1 for details of the receive buffer descriptor list.

Table 32-1. Receive Buffer Descriptor Entry

Bit	Function
	Word 0
31:2	Address of beginning of buffer
1	Wrap - marks last descriptor in receive buffer descriptor list.
0	Ownership - needs to be zero for the EMAC to write data to the receive buffer. The EMAC sets this to one once it has successfully written a frame to memory. Software has to clear this bit before the buffer can be used again.
	Word 1
31	Global all ones broadcast address detected
30	Multicast hash match
29	Unicast hash match
28	External address match
27	Reserved for future use

Table 32-1. Receive Buffer Descriptor Entry (Continued)

Bit	Function
26	Specific address register 1 match
25	Specific address register 2 match
24	Specific address register 3 match
23	Specific address register 4 match
22	Type ID match
21	VLAN tag detected (i.e., type id of 0x8100)
20	Priority tag detected (i.e., type id of 0x8100 and null VLAN identifier)
19:17	VLAN priority (only valid if bit 21 is set)
16	Concatenation format indicator (CFI) bit (only valid if bit 21 is set)
15	End of frame - when set the buffer contains the end of a frame. If end of frame is not set, then the only other valid status are bits 12, 13 and 14.
14	Start of frame - when set the buffer contains the start of a frame. If both bits 15 and 14 are set, then the buffer contains a whole frame.
13:12	Receive buffer offset - indicates the number of bytes by which the data in the first buffer is offset from the word address. Updated with the current values of the network configuration register. If jumbo frame mode is enabled through bit 3 of the network configuration register, then bits 13:12 of the receive buffer descriptor entry are used to indicate bits 13:12 of the frame length.
11:0	Length of frame including FCS (if selected). Bits 13:12 are also used if jumbo frame mode is selected.

To receive frames, the buffer descriptors must be initialized by writing an appropriate address to bits 31 to 2 in the first word of each list entry. Bit zero must be written with zero. Bit one is the wrap bit and indicates the last entry in the list.

The start location of the receive buffer descriptor list must be written to the receive buffer queue pointer register before setting the receive enable bit in the network control register to enable receive. As soon as the receive block starts writing received frame data to the receive FIFO, the receive buffer manager reads the first receive buffer location pointed to by the receive buffer queue pointer register.

If the filter block then indicates that the frame should be copied to memory, the receive data DMA operation starts writing data into the receive buffer. If an error occurs, the buffer is recovered. If the current buffer pointer has its wrap bit set or is the 1024th descriptor, the next receive buffer location is read from the beginning of the receive descriptor list. Otherwise, the next receive buffer location is read from the next word in memory.

There is an 11-bit counter to count out the 2048 word locations of a maximum length, receive buffer descriptor list. This is added with the value originally written to the receive buffer queue pointer register to produce a pointer into the list. A read of the receive buffer queue pointer register returns the pointer value, which is the queue entry currently being accessed. The counter is reset after receive status is written to a descriptor that has its wrap bit set or rolls over to zero after 1024 descriptors have been accessed. The value written to the receive buffer pointer register may be any word-aligned address, provided that there are at least 2048 word locations available between the pointer and the top of the memory.

Section 3.6 of the AMBA 2.0 specification states that bursts should not cross 1K boundaries. As receive buffer manager writes are bursts of two words, to ensure that this does not occur, it is best to write the pointer register with the least three significant bits set to zero. As receive buffers are used, the receive buffer manager sets bit zero of the first word of the descriptor to indicate *used*. If a receive error is detected the receive buffer currently being written is recovered. Previous buffers are not recovered. Software should search through the *used* bits in the buffer descriptors to find out how many frames have been received. It should be checking the start-of-frame and end-of-frame bits, and not rely on the value returned by the receive buffer queue pointer register which changes continuously as more buffers are used.

For CRC errored frames, excessive length frames or length field mismatched frames, all of which are counted in the statistics registers, it is possible that a frame fragment might be stored in a sequence of receive buffers. Software can detect this by looking for start of frame bit set in a buffer following a buffer with no end of frame bit set.

For a properly working Ethernet system, there should be no excessively long frames or frames greater than 128 bytes with CRC/FCS errors. Collision fragments are less than 128 bytes long. Therefore, it is a rare occurrence to find a frame fragment in a receive buffer.

If bit zero is set when the receive buffer manager reads the location of the receive buffer, then the buffer has already been used and cannot be used again until software has processed the frame and cleared bit zero. In this case, the DMA block sets the buffer not available bit in the receive status register and triggers an interrupt.

If bit zero is set when the receive buffer manager reads the location of the receive buffer and a frame is being received, the frame is discarded and the receive resource error statistics register is incremented.

A receive overrun condition occurs when bus was not granted in time or because HRESP was not OK (bus error). In a receive overrun condition, the receive overrun interrupt is asserted and the buffer currently being written is recovered. The next frame received with an address that is recognized reuses the buffer.

If bit 17 of the network configuration register is set, the FCS of received frames shall not be copied to memory. The frame length indicated in the receive status field shall be reduced by four bytes in this case.

32.3.1.3 Transmit Buffer

Frames to be transmitted are stored in one or more transmit buffers. Transmit buffers can be between 0 and 2047 bytes long, so it is possible to transmit frames longer than the maximum length specified in IEEE Standard 802.3. Zero length buffers are allowed. The maximum number of buffers permitted for each transmit frame is 128.

The start location for each transmit buffer is stored in memory in a list of transmit buffer descriptors at a location pointed to by the transmit buffer queue pointer register. Each list entry consists of two words, the first being the byte address of the transmit buffer and the second containing the transmit control and status. Frames can be transmitted with or without automatic CRC generation. If CRC is automatically generated, pad is also automatically generated to take frames to a minimum length of 64 bytes. Table 32-2 on page 615 defines an entry in the transmit buffer descriptor list. To transmit frames, the buffer descriptors must be initialized by writing an appropriate byte address to bits 31 to 0 in the first word of each list entry. The second transmit buffer descriptor is initialized with control information that indicates the length of the buffer, whether or not it is to be transmitted with CRC and whether the buffer is the last buffer in the frame.

After transmission, the control bits are written back to the second word of the first buffer along with the "used" bit and other status information. Bit 31 is the "used" bit which must be zero when the control word is read if transmission is to happen. It is written to one when a frame has been transmitted. Bits 27, 28 and 29 indicate various transmit error conditions. Bit 30 is the “wrap” bit which can be set for any buffer within a frame. If no wrap bit is encountered after 1024 descriptors, the queue pointer rolls over to the start in a similar fashion to the receive queue.

The transmit buffer queue pointer register must not be written while transmit is active. If a new value is written to the transmit buffer queue pointer register, the queue pointer resets itself to point to the beginning of the new queue. If transmit is disabled by writing to bit 3 of the network control, the transmit buffer queue pointer register resets to point to the beginning of the transmit queue. Note that disabling receive does not have the same effect on the receive queue pointer.

Once the transmit queue is initialized, transmit is activated by writing to bit 9, the *Transmit Start* bit of the network control register. Transmit is halted when a buffer descriptor with its *used* bit set is read, or if a transmit error occurs, or by writing to the transmit halt bit of the network control register. (Transmission is suspended if a pause frame is received while the pause enable bit is set in the network configuration register.) Rewriting the start bit while transmission is active is allowed.

Transmission control is implemented with a Tx_go variable which is readable in the transmit status register at bit location 3. The Tx_go variable is reset when:

- transmit is disabled
- a buffer descriptor with its ownership bit set is read
- a new value is written to the transmit buffer queue pointer register
- bit 10, tx_halt, of the network control register is written
- there is a transmit error such as too many retries or a transmit underrun.

To set tx_go, write to bit 9, tx_start, of the network control register. Transmit halt does not take effect until any ongoing transmit finishes. If a collision occurs during transmission of a multi-buffer frame, transmission automatically restarts from the first buffer of the frame. If a “used” bit is read midway through transmission of a multi-buffer frame, this is treated as a transmit error. Transmission stops, tx_er is asserted and the FCS is bad.

If transmission stops due to a transmit error, the transmit queue pointer resets to point to the beginning of the transmit queue. Software needs to re-initialize the transmit queue after a transmit error.

If transmission stops due to a “used” bit being read at the start of the frame, the transmission queue pointer is not reset and transmit starts from the same transmit buffer descriptor when the transmit start bit is written

Table 32-2. Transmit Buffer Descriptor Entry

Bit	Function
	Word 0
31:0	**Byte Address of buffer**
	Word 1
31	Used. Needs to be zero for the EMAC to read data from the transmit buffer. The EMAC sets this to one for the first buffer of a frame once it has been successfully transmitted. Software has to clear this bit before the buffer can be used again. Note: This bit is only set for the first buffer in a frame unlike receive where all buffers have the Used bit set once used.

Table 32-2. Transmit Buffer Descriptor Entry

Bit	Function
30	Wrap. Marks last descriptor in transmit buffer descriptor list.
29	Retry limit exceeded, transmit error detected
28	Transmit underrun, occurs either when hresp is not OK (bus error) or the transmit data could not be fetched in time or when buffers are exhausted in mid frame.
27	Buffers exhausted in mid frame
26:17	Reserved
16	No CRC. When set, no CRC is appended to the current frame. This bit only needs to be set for the last buffer of a frame.
15	Last buffer. When set, this bit indicates the last buffer in the current frame has been reached.
14:11	Reserved
10:0	Length of buffer

32.3.2 Transmit Block

This block transmits frames in accordance with the Ethernet IEEE 802.3 CSMA/CD protocol. Frame assembly starts by adding preamble and the start frame delimiter. Data is taken from the transmit FIFO a word at a time. Data is transmitted least significant nibble first. If necessary, padding is added to increase the frame length to 60 bytes. CRC is calculated as a 32-bit polynomial. This is inverted and appended to the end of the frame, taking the frame length to a minimum of 64 bytes. If the No CRC bit is set in the second word of the last buffer descriptor of a transmit frame, neither pad nor CRC are appended.

In full-duplex mode, frames are transmitted immediately. Back-to-back frames are transmitted at least 96 bit times apart to guarantee the interframe gap.

In half-duplex mode, the transmitter checks carrier sense. If asserted, it waits for it to de-assert and then starts transmission after the interframe gap of 96 bit times. If the collision signal is asserted during transmission, the transmitter transmits a jam sequence of 32 bits taken from the data register and then retry transmission after the back off time has elapsed.

The back-off time is based on an XOR of the 10 least significant bits of the data coming from the transmit FIFO and a 10-bit pseudo random number generator. The number of bits used depends on the number of collisions seen. After the first collision, 1 bit is used, after the second 2, and so on up to 10. Above 10, all 10 bits are used. An error is indicated and no further attempts are made if 16 attempts cause collisions.

If transmit DMA underruns, bad CRC is automatically appended using the same mechanism as jam insertion and the tx_er signal is asserted. For a properly configured system, this should never happen.

If the back pressure bit is set in the network control register in half duplex mode, the transmit block transmits 64 bits of data, which can consist of 16 nibbles of 1011 or in bit-rate mode 64 1s, whenever it sees an incoming frame to force a collision. This provides a way of implementing flow control in half-duplex mode.

32.3.3 Pause Frame Support

The start of an 802.3 pause frame is as follows:

Table 32-3. Start of an 802.3 Pause Frame

Destination Address	Source Address	Type (Mac Control Frame)	Pause Opcode	Pause Time
0x0180C2000001	6 bytes	0x8808	0x0001	2 bytes

The network configuration register contains a receive pause enable bit (13). If a valid pause frame is received, the pause time register is updated with the frame's pause time, regardless of its current contents and regardless of the state of the configuration register bit 13. An interrupt (12) is triggered when a pause frame is received, assuming it is enabled in the interrupt mask register. If bit 13 is set in the network configuration register and the value of the pause time register is non-zero, no new frame is transmitted until the pause time register has decremented to zero.

The loading of a new pause time, and hence the pausing of transmission, only occurs when the EMAC is configured for full-duplex operation. If the EMAC is configured for half-duplex, there is no transmission pause, but the pause frame received interrupt is still triggered.

A valid pause frame is defined as having a destination address that matches either the address stored in specific address register 1 or matches 0x0180C2000001 and has the MAC control frame type ID of 0x8808 and the pause opcode of 0x0001. Pause frames that have FCS or other errors are treated as invalid and are discarded. Valid pause frames received increment the Pause Frame Received statistic register.

The pause time register decrements every 512 bit times (i.e., 128 `rx_clks` in nibble mode) once transmission has stopped. For test purposes, the register decrements every `rx_clk` cycle once transmission has stopped if bit 12 (retry test) is set in the network configuration register. If the pause enable bit (13) is not set in the network configuration register, then the decrementing occurs regardless of whether transmission has stopped or not.

An interrupt (13) is asserted whenever the pause time register decrements to zero (assuming it is enabled in the interrupt mask register).

32.3.4 Receive Block

The receive block checks for valid preamble, FCS, alignment and length, presents received frames to the DMA block and stores the frames destination address for use by the address checking block. If, during frame reception, the frame is found to be too long or rx_er is asserted, a bad frame indication is sent to the DMA block. The DMA block then ceases sending data to memory. At the end of frame reception, the receive block indicates to the DMA block whether the frame is good or bad. The DMA block recovers the current receive buffer if the frame was bad. The receive block signals the register block to increment the alignment error, the CRC (FCS) error, the short frame, long frame, jabber error, the receive symbol error statistics and the length field mismatch statistics.

The enable bit for jumbo frames in the network configuration register allows the EMAC to receive jumbo frames of up to 10240 bytes in size. This operation does not form part of the IEEE802.3 specification and is disabled by default. When jumbo frames are enabled, frames received with a frame size greater than 10240 bytes are discarded.

32.3.5 Address Checking Block

The address checking (or filter) block indicates to the DMA block which receive frames should be copied to memory. Whether a frame is copied depends on what is enabled in the network configuration register, the state of the external match pin, the contents of the specific address and hash registers and the frame's destination address. In this implementation of the EMAC, the frame's source address is not checked. Provided that bit 18 of the Network Configuration register is not set, a frame is not copied to memory if the EMAC is transmitting in half duplex mode at the time a destination address is received. If bit 18 of the Network Configuration register is set, frames can be received while transmitting in half-duplex mode.

Ethernet frames are transmitted a byte at a time, least significant bit first. The first six bytes (48 bits) of an Ethernet frame make up the destination address. The first bit of the destination address, the LSB of the first byte of the frame, is the group/individual bit: this is *One* for multicast addresses and *Zero* for unicast. The *All Ones* address is the broadcast address, and a special case of multicast.

The EMAC supports recognition of four specific addresses. Each specific address requires two registers, specific address register bottom and specific address register top. Specific address register bottom stores the first four bytes of the destination address and specific address register top contains the last two bytes. The addresses stored can be specific, group, local or universal.

The destination address of received frames is compared against the data stored in the specific address registers once they have been activated. The addresses are deactivated at reset or when their corresponding specific address register bottom is written. They are activated when specific address register top is written. If a receive frame address matches an active address, the frame is copied to memory.

The following example illustrates the use of the address match registers for a MAC address of 21:43:65:87:A9:CB.

Preamble 55

SFD D5

DA (Octet0 - LSB) 21

DA(Octet 1) 43

DA(Octet 2) 65

DA(Octet 3) 87

DA(Octet 4) A9

DA (Octet5 - MSB) CB

SA (LSB) 00

SA 00

SA 00

SA 00

SA 00

SA (MSB) 43

SA (LSB) 21

The sequence above shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom as shown. For a successful match to specific address 1, the following address matching registers must be set up:

- Base address + 0x98 0x87654321 (Bottom)
- Base address + 0x9C 0x0000CBA9 (Top)

And for a successful match to the Type ID register, the following should be set up:

- Base address + 0xB8 0x00004321

32.3.6 Broadcast Address

The broadcast address of 0xFFFFFFFFFFFF is recognized if the 'no broadcast' bit in the network configuration register is zero.

32.3.7 Hash Addressing

The hash address register is 64 bits long and takes up two locations in the memory map. The least significant bits are stored in hash register bottom and the most significant bits in hash register top.

The unicast hash enable and the multicast hash enable bits in the network configuration register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 64-bit hash register using the following hash function. The hash function is an *exclusive or* of every sixth bit of the destination address.

hash_index[5] = da[5] ^ da[11] ^ da[17] ^ da[23] ^ da[29] ^ da[35] ^ da[41] ^ da[47]

hash_index[4] = da[4] ^ da[10] ^ da[16] ^ da[22] ^ da[28] ^ da[34] ^ da[40] ^ da[46]

hash_index[3] = da[3] ^ da[09] ^ da[15] ^ da[21] ^ da[27] ^ da[33] ^ da[39] ^ da[45]

hash_index[2] = da[2] ^ da[08] ^ da[14] ^ da[20] ^ da[26] ^ da[32] ^ da[38] ^ da[44]

hash_index[1] = da[1] ^ da[07] ^ da[13] ^ da[19] ^ da[25] ^ da[31] ^ da[37] ^ da[43]

hash_index[0] = da[0] ^ da[06] ^ da[12] ^ da[18] ^ da[24] ^ da[30] ^ da[36] ^ da[42]

`da[0]` represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and `da[47]` represents the most significant bit of the last byte received.

If the hash index points to a bit that is set in the hash register, then the frame is matched according to whether the frame is multicast or unicast.

A multicast match is signalled if the multicast hash enable bit is set. da[0] is 1 and the hash index points to a bit set in the hash register.

A unicast match is signalled if the unicast hash enable bit is set. da[0] is 0 and the hash index points to a bit set in the hash register.

To receive all multicast frames, the hash register should be set with all ones and the multicast hash enable bit should be set in the network configuration register.

32.3.8 Copy All Frames (or Promiscuous Mode)

If the copy all frames bit is set in the network configuration register, then all non-errored frames are copied to memory. For example, frames that are too long, too short, or have FCS errors or rx_er asserted during reception are discarded and all others are received. Frames with FCS errors are copied to memory if bit 19 in the network configuration register is set.

32.3.9 Type ID Checking

The contents of the type_id register are compared against the length/type ID of received frames (i.e., bytes 13 and 14). Bit 22 in the receive buffer descriptor status is set if there is a match. The reset state of this register is zero which is unlikely to match the length/type ID of any valid Ethernet frame.

Note: A type ID match does not affect whether a frame is copied to memory.

32.3.10 VLAN Support

An Ethernet encoded 802.1Q VLAN tag looks like this:

Table 32-4. 802.1Q VLAN Tag

TPID (Tag Protocol Identifier) 16 bits	TCI (Tag Control Information) 16 bits
0x8100	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the 13^{th} byte of the frame, adding an extra four bytes to the frame. If the VID (VLAN identifier) is null (0x000), this indicates a priority-tagged frame. The MAC can support frame lengths up to 1536 bytes, 18 bytes more than the original Ethernet maximum frame length of 1518 bytes. This is achieved by setting bit 8 in the network configuration register.

The following bits in the receive buffer descriptor status word give information about VLAN tagged frames:

- Bit 21 set if receive frame is VLAN tagged (i.e. type id of 0x8100)
- Bit 20 set if receive frame is priority tagged (i.e. type id of 0x8100 and null VID). (If bit 20 is set bit 21 is set also.)
- Bit 19, 18 and 17 set to priority if bit 21 is set
- Bit 16 set to CFI if bit 21 is set

32.3.11 PHY Maintenance

The register EMAC_MAN enables the EMAC to communicate with a PHY by means of the MDIO interface. It is used during auto-negotiation to ensure that the EMAC and the PHY are configured for the same speed and duplex configuration.

The PHY maintenance register is implemented as a shift register. Writing to the register starts a shift operation which is signalled as complete when bit two is set in the network status register (about 2000 MCK cycles later when bit ten is set to zero, and bit eleven is set to one in the network configuration register). An interrupt is generated as this bit is set. During this time, the MSB of the register is output on the MDIO pin and the LSB updated from the MDIO pin with each MDC cycle. This causes transmission of a PHY management frame on MDIO.

Reading during the shift operation returns the current contents of the shift register. At the end of management operation, the bits have shifted back to their original locations. For a read operation, the data bits are updated with data read from the PHY. It is important to write the correct values to the register to ensure a valid PHY management frame is produced.

The MDIO interface can read IEEE 802.3 clause 45 PHYs as well as clause 22 PHYs. To read clause 45 PHYs, bits[31:28] should be written as 0x0011. For a description of MDC generation, see the network configuration register in the "Network Control Register" on page 627.

32.3.12 Media Independent Interface

The Ethernet MAC is capable of interfacing to both RMII and MII Interfaces. The RMII bit in the EMAC_USRIO register controls the interface that is selected. When this bit is set, the RMII interface is selected, else the MII interface is selected.

The MII and RMII interface are capable of both 10Mb/s and 100Mb/s data rates as described in the IEEE 802.3u standard. The signals used by the MII and RMII interfaces are described in Table 32-5.

Table 32-5. Pin Configuration

Pin Name	MII	RMII
ETXCK_EREFCK	ETXCK: Transmit Clock	EREFCK: Reference Clock
ECRS	ECRS: Carrier Sense	
ECOL	ECOL: Collision Detect	
ERXDV	ERXDV: Data Valid	ECRSDV: Carrier Sense/Data Valid
ERX0 - ERX3	ERX0 - ERX3: 4-bit Receive Data	ERX0 - ERX1: 2-bit Receive Data
ERXER	ERXER: Receive Error	ERXER: Receive Error
ERXCK	ERXCK: Receive Clock	
ETXEN	ETXEN: Transmit Enable	ETXEN: Transmit Enable
ETX0-ETX3	ETX0 - ETX3: 4-bit Transmit Data	ETX0 - ETX1: 2-bit Transmit Data
ETXER	ETXER: Transmit Error	

The intent of the RMII is to provide a reduced pin count alternative to the IEEE 802.3u MII. It uses 2 bits for transmit (ETX0 and ETX1) and two bits for receive (ERX0 and ERX1). There is a Transmit Enable (ETXEN), a Receive Error (ERXER), a Carrier Sense (ECRS_DV), and a 50 MHz Reference Clock (ETXCK_EREFCK) for 100Mb/s data rate.

32.3.12.1 RMII Transmit and Receive Operation

The same signals are used internally for both the RMII and the MII operations. The RMII maps these signals in a more pin-efficient manner. The transmit and receive bits are converted from a 4-bit parallel format to a 2-bit parallel scheme that is clocked at twice the rate. The carrier sense and data valid signals are combined into the ECRSDV signal. This signal contains information on carrier sense, FIFO status, and validity of the data. Transmit error bit (ETXER) and collision detect (ECOL) are not used in RMII mode.

32.4 Programming Interface

32.4.1 Initialization

32.4.1.1 Configuration

Initialization of the EMAC configuration (e.g., loop-back mode, frequency ratios) must be done while the transmit and receive circuits are disabled. See the description of the network control register and network configuration register earlier in this document.

To change loop-back mode, the following sequence of operations must be followed:

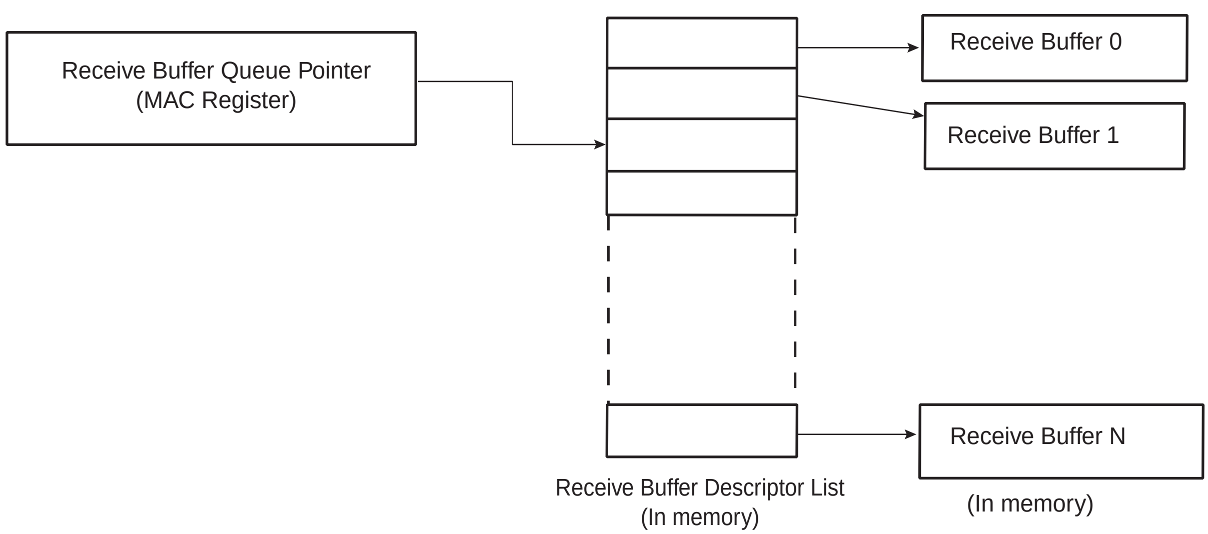
1. Write to network control register to disable transmit and receive circuits.
2. Write to network control register to change loop-back mode.
3. Write to network control register to re-enable transmit or receive circuits.

Note: These writes to network control register cannot be combined in any way.

32.4.1.2 Receive Buffer List

Receive data is written to areas of data (i.e., buffers) in system memory. These buffers are listed in another data structure that also resides in main memory. This data structure (receive buffer queue) is a sequence of descriptor entries as defined in "Receive Buffer Descriptor Entry" on page 612. It points to this data structure.

Figure 32-2. Receive Buffer List



To create the list of buffers:

1. Allocate a number (*n*) of buffers of 128 bytes in system memory.
2. Allocate an area 2*n* words for the receive buffer descriptor entry in system memory and create *n* entries in this list. Mark all entries in this list as owned by EMAC, i.e., bit 0 of word 0 set to 0.
3. If less than 1024 buffers are defined, the last descriptor must be marked with the wrap bit (bit 1 in word 0 set to 1).
4. Write address of receive buffer descriptor entry to EMAC register `receive_buffer` queue pointer.
5. The receive circuits can then be enabled by writing to the address recognition registers and then to the network control register.

32.4.1.3 Transmit Buffer List

Transmit data is read from areas of data (the buffers) in system memory These buffers are listed in another data structure that also resides in main memory. This data structure (Transmit Buffer Queue) is a sequence of descriptor entries (as defined in Table 32-2 on page 615) that points to this data structure.

To create this list of buffers:

1. Allocate a number (*n*) of buffers of between 1 and 2047 bytes of data to be transmitted in system memory. Up to 128 buffers per frame are allowed.
2. Allocate an area 2*n* words for the transmit buffer descriptor entry in system memory and create N entries in this list. Mark all entries in this list as owned by EMAC, i.e. bit 31 of word 1 set to 0.
3. If fewer than 1024 buffers are defined, the last descriptor must be marked with the wrap bit — bit 30 in word 1 set to 1.
4. Write address of transmit buffer descriptor entry to EMAC register transmit_buffer queue pointer.
5. The transmit circuits can then be enabled by writing to the network control register.

32.4.1.4 Address Matching

The EMAC register-pair hash address and the four specific address register-pairs must be written with the required values. Each register-pair comprises a bottom register and top register, with the bottom register being written first. The address matching is disabled for a particular register-pair after the bottom-register has been written and re-enabled when the top register is written. See “Address Checking Block” on page 618. for details of address matching. Each register-pair may be written at any time, regardless of whether the receive circuits are enabled or disabled.

32.4.1.5 Interrupts

There are 14 interrupt conditions that are detected within the EMAC. These are ORed to make a single interrupt. Depending on the overall system design, this may be passed through a further level of interrupt collection (interrupt controller). On receipt of the interrupt signal, the CPU enters the interrupt handler (Refer to the AIC programmer datasheet). To ascertain which interrupt has been generated, read the interrupt status register. Note that this register clears itself when read. At reset, all interrupts are disabled. To enable an interrupt, write to interrupt enable register with the pertinent interrupt bit set to 1. To disable an interrupt, write to interrupt disable register with the pertinent interrupt bit set to 1. To check whether an interrupt is enabled or disabled, read interrupt mask register: if the bit is set to 1, the interrupt is disabled.

32.4.1.6 Transmitting Frames

To set up a frame for transmission:

1. Enable transmit in the network control register.
2. Allocate an area of system memory for transmit data. This does not have to be contiguous, varying byte lengths can be used as long as they conclude on byte borders.
3. Set-up the transmit buffer list.
4. Set the network control register to enable transmission and enable interrupts.
5. Write data for transmission into these buffers.
6. Write the address to transmit buffer descriptor queue pointer.
7. Write control and length to word one of the transmit buffer descriptor entry.

8. Write to the transmit start bit in the network control register.

32.4.1.7 Receiving Frames

When a frame is received and the receive circuits are enabled, the EMAC checks the address and, in the following cases, the frame is written to system memory:

- if it matches one of the four specific address registers.
- if it matches the hash address function.
- if it is a broadcast address (0xFFFFFFFFFFFF) and broadcasts are allowed.
- if the EMAC is configured to copy all frames.

The register receive buffer queue pointer points to the next entry (see Table 32-1 on page 612) and the EMAC uses this as the address in system memory to write the frame to. Once the frame has been completely and successfully received and written to system memory, the EMAC then updates the receive buffer descriptor entry with the reason for the address match and marks the area as being owned by software. Once this is complete an interrupt receive complete is set. Software is then responsible for handling the data in the buffer and then releasing the buffer by writing the ownership bit back to 0.

If the EMAC is unable to write the data at a rate to match the incoming frame, then an interrupt receive overrun is set. If there is no receive buffer available, i.e., the next buffer is still owned by software, the interrupt receive buffer not available is set. If the frame is not successfully received, a statistic register is incremented and the frame is discarded without informing software.

32.5 Ethernet MAC 10/100 (EMAC) User Interface

Table 32-6. Ethernet MAC 10/100 (EMAC) Register Mapping

Offset	Register	Name	Access	Reset Value
0x00	Network Control Register	EMAC_NCR	Read/Write	0
0x04	Network Configuration Register	EMAC_NCFG	Read/Write	0x800
0x08	Network Status Register	EMAC_NSR	Read-only	-
0x0C	Reserved			
0x10	Reserved			
0x14	Transmit Status Register	EMAC_TSR	Read/Write	0x0000_0000
0x18	Receive Buffer Queue Pointer Register	EMAC_RBQP	Read/Write	0x0000_0000
0x1C	Transmit Buffer Queue Pointer Register	EMAC_TBQP	Read/Write	0x0000_0000
0x20	Receive Status Register	EMAC_RSR	Read/Write	0x0000_0000
0x24	Interrupt Status Register	EMAC_ISR	Read/Write	0x0000_0000
0x28	Interrupt Enable Register	EMAC_IER	Write-only	-
0x2C	Interrupt Disable Register	EMAC_IDR	Write-only	-
0x30	Interrupt Mask Register	EMAC_IMR	Read-only	0x0000_3FFF
0x34	Phy Maintenance Register	EMAC_MAN	Read/Write	0x0000_0000
0x38	Pause Time Register	EMAC_PTR	Read/Write	0x0000_0000
0x3C	Pause Frames Received Register	EMAC_PFR	Read/Write	0x0000_0000
0x40	Frames Transmitted Ok Register	EMAC_FTO	Read/Write	0x0000_0000
0x44	Single Collision Frames Register	EMAC_SCF	Read/Write	0x0000_0000
0x48	Multiple Collision Frames Register	EMAC_MCF	Read/Write	0x0000_0000
0x4C	Frames Received Ok Register	EMAC_FRO	Read/Write	0x0000_0000
0x50	Frame Check Sequence Errors Register	EMAC_FCSE	Read/Write	0x0000_0000
0x54	Alignment Errors Register	EMAC_ALE	Read/Write	0x0000_0000
0x58	Deferred Transmission Frames Register	EMAC_DTF	Read/Write	0x0000_0000
0x5C	Late Collisions Register	EMAC_LCOL	Read/Write	0x0000_0000
0x60	Excessive Collisions Register	EMAC_ECOL	Read/Write	0x0000_0000
0x64	Transmit Underrun Errors Register	EMAC_TUND	Read/Write	0x0000_0000
0x68	Carrier Sense Errors Register	EMAC_CSE	Read/Write	0x0000_0000
0x6C	Receive Resource Errors Register	EMAC_RRE	Read/Write	0x0000_0000
0x70	Receive Overrun Errors Register	EMAC_ROV	Read/Write	0x0000_0000
0x74	Receive Symbol Errors Register	EMAC_RSE	Read/Write	0x0000_0000
0x78	Excessive Length Errors Register	EMAC_ELE	Read/Write	0x0000_0000
0x7C	Receive Jabbers Register	EMAC_RJA	Read/Write	0x0000_0000
0x80	Undersize Frames Register	EMAC_USF	Read/Write	0x0000_0000
0x84	SQE Test Errors Register	EMAC_STE	Read/Write	0x0000_0000
0x88	Received Length Field Mismatch Register	EMAC_RLE	Read/Write	0x0000_0000

Table 32-6. Ethernet MAC 10/100 (EMAC) Register Mapping (Continued)

Offset	Register	Name	Access	Reset Value
0x90	Hash Register Bottom [31:0] Register	EMAC_HRB	Read/Write	0x0000_0000
0x94	Hash Register Top [63:32] Register	EMAC_HRT	Read/Write	0x0000_0000
0x98	Specific Address 1 Bottom Register	EMAC_SA1B	Read/Write	0x0000_0000
0x9C	Specific Address 1 Top Register	EMAC_SA1T	Read/Write	0x0000_0000
0xA0	Specific Address 2 Bottom Register	EMAC_SA2B	Read/Write	0x0000_0000
0xA4	Specific Address 2 Top Register	EMAC_SA2T	Read/Write	0x0000_0000
0xA8	Specific Address 3 Bottom Register	EMAC_SA3B	Read/Write	0x0000_0000
0xAC	Specific Address 3 Top Register	EMAC_SA3T	Read/Write	0x0000_0000
0xB0	Specific Address 4 Bottom Register	EMAC_SA4B	Read/Write	0x0000_0000
0xB4	Specific Address 4 Top Register	EMAC_SA4T	Read/Write	0x0000_0000
0xB8	Type ID Checking Register	EMAC_TID	Read/Write	0x0000_0000
0xC0	User Input/Output Register	EMAC_USRIO	Read/Write	0x0000_0000
0xC8 - 0xFC	Reserved	–	–	–

32.5.1 Network Control Register

Register Name: EMAC_NCR

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
–	–	–	–	–	THALT	TSTART	BP

7	6	5	4	3	2	1	0
WESTAT	INCSTAT	CLRSTAT	MPE	TE	RE	LLB	LB

- **LB: LoopBack**

Asserts the loopback signal to the PHY.

- **LLB: Loopback local**

Connects `txd` to `rxd`, `tx_en` to `rx_dv`, forces full duplex and drives `rx_clk` and `tx_clk` with `pclk` divided by 4. `rx_clk` and `tx_clk` may glitch as the EMAC is switched into and out of internal loop back. It is important that receive and transmit circuits have already been disabled when making the switch into and out of internal loop back.

- **RE: Receive enable**

When set, enables the EMAC to receive data. When reset, frame reception stops immediately and the receive FIFO is cleared. The receive queue pointer register is unaffected.

- **TE: Transmit enable**

When set, enables the Ethernet transmitter to send data. When reset transmission, stops immediately, the transmit FIFO and control registers are cleared and the transmit queue pointer register resets to point to the start of the transmit descriptor list.

- **MPE: Management port enable**

Set to one to enable the management port. When zero, forces MDIO to high impedance state and MDC low.

- **CLRSTAT: Clear statistics registers**

This bit is write only. Writing a one clears the statistics registers.

- **INCSTAT: Increment statistics registers**

This bit is write only. Writing a one increments all the statistics registers by one for test purposes.

- **WESTAT: Write enable for statistics registers**

Setting this bit to one makes the statistics registers writable for functional test purposes.

- **BP: Back pressure**

If set in half duplex mode, forces collisions on all received frames.

- **TSTART: Start transmission**

Writing one to this bit starts transmission.

- **THALT: Transmit halt**

Writing one to this bit halts transmission as soon as any ongoing frame transmission ends.

32.5.2 Network Configuration Register

Register Name: EMAC_NCFGR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
–	–	–	–	IRXFCS	EFRHD	DRFCS	RLCE

15	14	13	12	11	10	9	8
RBOF		PAE	RTY	CLK		–	BIG

7	6	5	4	3	2	1	0
UNI	MTI	NBC	CAF	JFRAME	–	FD	SPD

- **SPD: Speed**

Set to 1 to indicate 100 Mbit/s operation, 0 for 10 Mbit/s. The value of this pin is reflected on the `speed` pin.

- **FD: Full Duplex**

If set to 1, the transmit block ignores the state of collision and carrier sense and allows receive while transmitting. Also controls the `half_duplex` pin.

- **CAF: Copy All Frames**

When set to 1, all valid frames are received.

- **JFRAME: Jumbo Frames**

Set to one to enable jumbo frames of up to 10240 bytes to be accepted.

- **NBC: No Broadcast**

When set to 1, frames addressed to the broadcast address of all ones are not received.

- **MTI: Multicast Hash Enable**

When set, multicast frames are received when the 6-bit hash function of the destination address points to a bit that is set in the hash register.

- **UNI: Unicast Hash Enable**

When set, unicast frames are received when the 6-bit hash function of the destination address points to a bit that is set in the hash register.

- **BIG: Receive 1536 bytes frames**

Setting this bit means the EMAC receives frames up to 1536 bytes in length. Normally, the EMAC would reject any frame above 1518 bytes.

- **CLK: MDC clock divider**

Set according to system clock speed. This determines by what number system clock is divided to generate MDC. For conformance with 802.3, MDC must not exceed 2.5MHz (MDC is only active during MDIO read and write operations).

CLK	MDC
00	MCK divided by 8 (MCK up to 20 MHz)
01	MCK divided by 16 (MCK up to 40 MHz)
10	MCK divided by 32 (MCK up to 80 MHz)
11	MCK divided by 64 (MCK up to 160 MHz)

- **RTY: Retry test**

Must be set to zero for normal operation. If set to one, the back off between collisions is always one slot time. Setting this bit to one helps testing the too many retries condition. Also used in the pause frame tests to reduce the pause counters decrement time from 512 bit times, to every `rx_clk` cycle.

- **PAE: Pause Enable**

When set, transmission pauses when a valid pause frame is received.

- **RBOF: Receive Buffer Offset**

Indicates the number of bytes by which the received data is offset from the start of the first receive buffer.

RBOF	Offset
00	No offset from start of receive buffer
01	One-byte offset from start of receive buffer
10	Two-byte offset from start of receive buffer
11	Three-byte offset from start of receive buffer

- **RLCE: Receive Length field Checking Enable**

When set, frames with measured lengths shorter than their length fields are discarded. Frames containing a type ID in bytes 13 and 14 — length/type ID = 0600 — are not be counted as length errors.

- **DRFCS: Discard Receive FCS**

When set, the FCS field of received frames are not be copied to memory.

- **EFRHD:**

Enable Frames to be received in half-duplex mode while transmitting.

- **IRXFCS: Ignore RX FCS**

When set, frames with FCS/CRC errors are not rejected and no FCS error statistics are counted. For normal operation, this bit must be set to 0.

32.5.3 Network Status Register

Register Name: EMAC_NSR

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
–	–	–	–	–	IDLE	MDIO	–

- **MDIO**

Returns status of the mdio_in pin. Use the PHY maintenance register for reading managed frames rather than this bit.

- **IDLE**

0 = The PHY logic is running.

1 = The PHY management logic is idle (i.e., has completed).

32.5.4 Transmit Status Register

Register Name: EMAC_TSR

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
–	UND	COMP	BEX	TGO	RLE	COL	UBR

This register, when read, provides details of the status of a transmit. Once read, individual bits may be cleared by writing 1 to them. It is not possible to set a bit to 1 by writing to the register.

- **UBR: Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Cleared by writing a one to this bit.

- **COL: Collision Occurred**

Set by the assertion of collision. Cleared by writing a one to this bit.

- **RLE: Retry Limit exceeded**

Cleared by writing a one to this bit.

- **TGO: Transmit Go**

If high transmit is active.

- **BEX: Buffers exhausted mid frame**

If the buffers run out during transmission of a frame, then transmission stops, FCS shall be bad and tx_er asserted. Cleared by writing a one to this bit.

- **COMP: Transmit Complete**

Set when a frame has been transmitted. Cleared by writing a one to this bit.

- **UND: Transmit Underrun**

Set when transmit DMA was not able to read data from memory, either because the bus was not granted in time, because a not OK `hresp(bus error)` was returned or because a used bit was read midway through frame transmission. If this occurs, the transmitter forces `bad` CRC. Cleared by writing a one to this bit.

32.5.5 Receive Buffer Query Pointer Register

Register Name: EMAC_RBQP

Access Type: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR						–	–

This register points to the entry in the receive buffer queue (descriptor list) currently being used. It is written with the start location of the receive buffer descriptor list. The lower order bits increment as buffers are used up and wrap to their original values after either 1024 buffers or when the wrap bit of the entry is set.

Reading this register returns the location of the descriptor currently being accessed. This value increments as buffers are used. Software should not use this register for determining where to remove received frames from the queue as it constantly changes as new frames are received. Software should instead work its way through the buffer descriptor queue checking the used bits.

Receive buffer writes also comprise bursts of two words and, as with transmit buffer reads, it is recommended that bit 2 is always written with zero to prevent a burst crossing a 1K boundary, in violation of section 3.6 of the AMBA specification.

- **ADDR: Receive buffer queue pointer address**

Written with the address of the start of the receive queue, reads as a pointer to the current buffer being used.

32.5.6 Transmit Buffer Queue Pointer Register

Register Name: EMAC_TBQP

Access Type: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR						–	–

This register points to the entry in the transmit buffer queue (descriptor list) currently being used. It is written with the start location of the transmit buffer descriptor list. The lower order bits increment as buffers are used up and wrap to their original values after either 1024 buffers or when the wrap bit of the entry is set. This register can only be written when bit 3 in the transmit status register is low.

As transmit buffer reads consist of bursts of two words, it is recommended that bit 2 is always written with zero to prevent a burst crossing a 1K boundary, in violation of section 3.6 of the AMBA specification.

- **ADDR: Transmit buffer queue pointer address**

Written with the address of the start of the transmit queue, reads as a pointer to the first buffer of the frame being transmitted or about to be transmitted.

32.5.7 Receive Status Register

Register Name: EMAC_RSR

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
–	–	–	–	–	OVR	REC	BNA

This register, when read, provides details of the status of a receive. Once read, individual bits may be cleared by writing 1 to them. It is not possible to set a bit to 1 by writing to the register.

- **BNA: Buffer Not Available**

An attempt was made to get a new buffer and the pointer indicated that it was owned by the processor. The DMA rereads the pointer each time a new frame starts until a valid pointer is found. This bit is set at each attempt that fails even if it has not had a successful pointer read since it has been cleared.

Cleared by writing a one to this bit.

- **REC: Frame Received**

One or more frames have been received and placed in memory. Cleared by writing a one to this bit.

- **OVR: Receive Overrun**

The DMA block was unable to store the receive frame to memory, either because the bus was not granted in time or because a not OK `hresp(bus error)` was returned. The buffer is recovered if this happens.

Cleared by writing a one to this bit.

32.5.8 Interrupt Status Register

Register Name: EMAC_ISR

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
–	–	PTZ	PFR	HRESP	ROVR	–	–

7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame Done**

The PHY maintenance register has completed its operation. Cleared on read.

- **RCOMP: Receive Complete**

A frame has been stored in memory. Cleared on read.

- **RXUBR: Receive Used Bit Read**

Set when a receive buffer descriptor is read with its used bit set. Cleared on read.

- **TXUBR: Transmit Used Bit Read**

Set when a transmit buffer descriptor is read with its used bit set. Cleared on read.

- **TUND: Ethernet Transmit Buffer Underrun**

The transmit DMA did not fetch frame data in time for it to be transmitted or `hresp` returned not OK. Also set if a used bit is read mid-frame or when a new transmit queue pointer is written. Cleared on read.

- **RLE: Retry Limit Exceeded**

Cleared on read.

- **TXERR: Transmit Error**

Transmit buffers exhausted in mid-frame - transmit error. Cleared on read.

- **TCOMP: Transmit Complete**

Set when a frame has been transmitted. Cleared on read.

- **ROVR: Receive Overrun**

Set when the receive overrun status bit gets set. Cleared on read.

- **HRESP: Hresp not OK**

Set when the DMA block sees `a bus error`. Cleared on read.

- **PFR: Pause Frame Received**

Indicates a valid pause has been received. Cleared on a read.

- **PTZ: Pause Time Zero**

Set when the pause time register, 0x38 decrements to zero. Cleared on a read.

32.5.9 Interrupt Enable Register

Register Name: EMAC_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
–	–	PTZ	PFR	HRESP	ROVR	–	–

7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame sent**

Enable management done interrupt.

- **RCOMP: Receive Complete**

Enable receive complete interrupt.

- **RXUBR: Receive Used Bit Read**

Enable receive used bit read interrupt.

- **TXUBR: Transmit Used Bit Read**

Enable transmit used bit read interrupt.

- **TUND: Ethernet Transmit Buffer Underrun**

Enable transmit underrun interrupt.

- **RLE: Retry Limit Exceeded**

Enable retry limit exceeded interrupt.

- **TXERR**

Enable transmit buffers exhausted in mid-frame interrupt.

- **TCOMP: Transmit Complete**

Enable transmit complete interrupt.

- **ROVR: Receive Overrun**

Enable receive overrun interrupt.

- **HRESP: Hresp not OK**

Enable Hresp not OK interrupt.

- **PFR: Pause Frame Received**

Enable pause frame received interrupt.

- **PTZ: Pause Time Zero**

Enable pause time zero interrupt.

32.5.10 Interrupt Disable Register

Register Name: EMAC_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
–	–	PTZ	PFR	HRESP	ROVR	–	–

7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame sent**

Disable management done interrupt.

- **RCOMP: Receive Complete**

Disable receive complete interrupt.

- **RXUBR: Receive Used Bit Read**

Disable receive used bit read interrupt.

- **TXUBR: Transmit Used Bit Read**

Disable transmit used bit read interrupt.

- **TUND: Ethernet Transmit Buffer Underrun**

Disable transmit underrun interrupt.

- **RLE: Retry Limit Exceeded**

Disable retry limit exceeded interrupt.

- **TXERR**

Disable transmit buffers exhausted in mid-frame interrupt.

- **TCOMP: Transmit Complete**

Disable transmit complete interrupt.

- **ROVR: Receive Overrun**

Disable receive overrun interrupt.

- **HRESP: Hresp not OK**

Disable Hresp not OK interrupt.

- **PFR: Pause Frame Received**

Disable pause frame received interrupt.

- **PTZ: Pause Time Zero**

Disable pause time zero interrupt.

32.5.11 Interrupt Mask Register

Register Name: EMAC_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
–	–	PTZ	PFR	HRESP	ROVR	–	–

7	6	5	4	3	2	1	0
TCOMP	TXERR	RLE	TUND	TXUBR	RXUBR	RCOMP	MFD

- **MFD: Management Frame sent**

Management done interrupt masked.

- **RCOMP: Receive Complete**

Receive complete interrupt masked.

- **RXUBR: Receive Used Bit Read**

Receive used bit read interrupt masked.

- **TXUBR: Transmit Used Bit Read**

Transmit used bit read interrupt masked.

- **TUND: Ethernet Transmit Buffer Underrun**

Transmit underrun interrupt masked.

- **RLE: Retry Limit Exceeded**

Retry limit exceeded interrupt masked.

- **TXERR**

Transmit buffers exhausted in mid-frame interrupt masked.

- **TCOMP: Transmit Complete**

Transmit complete interrupt masked.

- **ROVR: Receive Overrun**

Receive overrun interrupt masked.

- **HRESP: Hresp not OK**

Hresp not OK interrupt masked.

- **PFR: Pause Frame Received**

Pause frame received interrupt masked.

- **PTZ: Pause Time Zero**

Pause time zero interrupt masked.

32.5.12 PHY Maintenance Register

Register Name: EMAC_MAN

Access Type: Read/Write

31	30	29	28	27	26	25	24
SOF		RW		PHYA			

23	22	21	20	19	18	17	16
PHYA	REGA					CODE	

15	14	13	12	11	10	9	8
DATA							

7	6	5	4	3	2	1	0
DATA							

- **DATA**

For a write operation this is written with the data to be written to the PHY.

After a read operation this contains the data read from the PHY.

- **CODE:**

Must be written to 10. Reads as written.

- **REGA: Register Address**

Specifies the register in the PHY to access.

- **PHYA: PHY Address**

- **RW: Read/Write**

10 is read; 01 is write. Any other value is an invalid PHY management frame

- **SOF: Start of frame**

Must be written 01 for a valid frame.

32.5.13 Pause Time Register

Register Name: EMAC_PTR

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
PTIME							

7	6	5	4	3	2	1	0
PTIME							

- **PTIME: Pause Time**

Stores the current value of the pause time register which is decremented every 512 bit times.

32.5.14 Hash Register Bottom

Register Name: EMAC_HRB

Access Type: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR:**

Bits 31:0 of the hash address register. See “Hash Addressing” on page 619.

32.5.15 Hash Register Top

Register Name: EMAC_HRT

Access Type: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR:**

Bits 63:32 of the hash address register. See “Hash Addressing” on page 619.

32.5.16 Specific Address 1 Bottom Register

Register Name: EMAC_SA1B

Access Type: Read/Write

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

- **ADDR**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

32.5.17 Specific Address 1 Top Register

Register Name: EMAC_SA1T

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**
ADDR							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADDR							

- **ADDR**

The most significant bits of the destination address, that is bits 47 to 32.

32.5.18 Specific Address 2 Bottom Register

Register Name: EMAC_SA2B

Access Type: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

32.5.19 Specific Address 2 Top Register

Register Name: EMAC_SA2T

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
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR**

The most significant bits of the destination address, that is bits 47 to 32.

32.5.20 Specific Address 3 Bottom Register

Register Name: EMAC_SA3B

Access Type: Read/Write

31	30	29	28	27	26	25	24
ADDR							

23	22	21	20	19	18	17	16
ADDR							

15	14	13	12	11	10	9	8
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

32.5.21 Specific Address 3 Top Register

Register Name: EMAC_SA3T

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
ADDR							

7	6	5	4	3	2	1	0
ADDR							

- **ADDR**

The most significant bits of the destination address, that is bits 47 to 32.

32.5.22 Specific Address 4 Bottom Register

Register Name: EMAC_SA4B

Access Type: Read/Write

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

- **ADDR**

Least significant bits of the destination address. Bit zero indicates whether the address is multicast or unicast and corresponds to the least significant bit of the first byte received.

32.5.23 Specific Address 4 Top Register

Register Name: EMAC_SA4T

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**
ADDR							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADDR							

- **ADDR**

The most significant bits of the destination address, that is bits 47 to 32.

32.5.24 Type ID Checking Register

Register Name: EMAC_TID

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
TID							

7	6	5	4	3	2	1	0
TID							

- **TID: Type ID checking**

For use in comparisons with received frames TypeID/Length field.

32.5.25 User Input/Output Register

Register Name: EMAC_USRIO

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**
–	–	–	–	–	–	CLKEN	RMII

- **RMII**

When set, this bit enables the RMII operation mode. When reset, it selects the MII mode.

- **CLKEN**

When set, this bit enables the transceiver input clock.

Setting this bit to 0 reduces power consumption when the treasurer is not used.

32.5.26 EMAC Statistic Registers

These registers reset to zero on a read and stick at all ones when they count to their maximum value. They should be read frequently enough to prevent loss of data. The receive statistics registers are only incremented when the receive enable bit is set in the network control register. To write to these registers, bit 7 must be set in the network control register. The statistics register block contains the following registers.

32.5.26.1 Pause Frames Received Register

Register Name: EMAC_PFR

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
FROK							

7	6	5	4	3	2	1	0
FROK							

- **FROK: Pause Frames received OK**

A 16-bit register counting the number of good pause frames received. A good frame has a length of 64 to 1518 (1536 if bit 8 set in network configuration register) and has no FCS, alignment or receive symbol errors.

32.5.26.2 Frames Transmitted OK Register

Register Name: EMAC_FTO

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
FTOK							

15	14	13	12	11	10	9	8
FTOK							

7	6	5	4	3	2	1	0
FTOK							

- **FTOK: Frames Transmitted OK**

A 24-bit register counting the number of frames successfully transmitted, i.e., no underrun and not too many retries.

32.5.26.3 Single Collision Frames Register

Register Name: EMAC_SCF

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
SCF							

7	6	5	4	3	2	1	0
SCF							

- **SCF: Single Collision Frames**

A 16-bit register counting the number of frames experiencing a single collision before being successfully transmitted, i.e., no underrun.

32.5.26.4 Multicollision Frames Register

Register Name: EMAC_MCF

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
MCF							

7	6	5	4	3	2	1	0
MCF							

- **MCF: Multicollision Frames**

A 16-bit register counting the number of frames experiencing between two and fifteen collisions prior to being successfully transmitted, i.e., no underrun and not too many retries.

32.5.26.5 Frames Received OK Register

Register Name: EMAC_FRO

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
FROK							

15	14	13	12	11	10	9	8
FROK							

7	6	5	4	3	2	1	0
FROK							

- **FROK: Frames Received OK**

A 24-bit register counting the number of good frames received, i.e., address recognized and successfully copied to memory. A good frame is of length 64 to 1518 bytes (1536 if bit 8 set in network configuration register) and has no FCS, alignment or receive symbol errors.

32.5.26.6 Frames Check Sequence Errors Register

Register Name: EMAC_FCSE

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
FCSE							

- **FCSE: Frame Check Sequence Errors**

An 8-bit register counting frames that are an integral number of bytes, have bad CRC and are between 64 and 1518 bytes in length (1536 if bit 8 set in network configuration register). This register is also incremented if a symbol error is detected and the frame is of valid length and has an integral number of bytes.

32.5.26.7 *Alignment Errors Register*

Register Name: EMAC_ALE

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
ALE							

- **ALE: Alignment Errors**

An 8-bit register counting frames that are not an integral number of bytes long and have bad CRC when their length is truncated to an integral number of bytes and are between 64 and 1518 bytes in length (1536 if bit 8 set in network configuration register). This register is also incremented if a symbol error is detected and the frame is of valid length and does not have an integral number of bytes.

32.5.26.8 *Deferred Transmission Frames Register*

Register Name: EMAC_DTF

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
DTF							

7	6	5	4	3	2	1	0
DTF							

- **DTF: Deferred Transmission Frames**

A 16-bit register counting the number of frames experiencing deferral due to carrier sense being active on their first attempt at transmission. Frames involved in any collision are not counted nor are frames that experienced a transmit underrun.

32.5.26.9 Late Collisions Register

Register Name: EMAC_LCOL

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
LCOL							

- **LCOL: Late Collisions**

An 8-bit register counting the number of frames that experience a collision after the slot time (512 bits) has expired. A late collision is counted twice; i.e., both as a collision and a late collision.

32.5.26.10 Excessive Collisions Register

Register Name: EMAC_EXCOL

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
EXCOL							

- **EXCOL: Excessive Collisions**

An 8-bit register counting the number of frames that failed to be transmitted because they experienced 16 collisions.

32.5.26.11 *Transmit Underrun Errors Register*

Register Name: EMAC_TUND

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
TUND							

- **TUND: Transmit Underruns**

An 8-bit register counting the number of frames not transmitted due to a transmit DMA underrun. If this register is incremented, then no other statistics register is incremented.

32.5.26.12 *Carrier Sense Errors Register*

Register Name: EMAC_CSE

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
CSE							

- **CSE: Carrier Sense Errors**

An 8-bit register counting the number of frames transmitted where carrier sense was not seen during transmission or where carrier sense was deasserted after being asserted in a transmit frame without collision (no underrun). Only incremented in half-duplex mode. The only effect of a carrier sense error is to increment this register. The behavior of the other statistics registers is unaffected by the detection of a carrier sense error.

32.5.26.13 Receive Resource Errors Register

Register Name: EMAC_RRE

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
RRE							

7	6	5	4	3	2	1	0
RRE							

- **RRE: Receive Resource Errors**

A 16-bit register counting the number of frames that were address matched but could not be copied to memory because no receive buffer was available.

32.5.26.14 Receive Overrun Errors Register

Register Name: EMAC_ROVR

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
ROVR							

- **ROVR: Receive Overrun**

An 8-bit register counting the number of frames that are address recognized but were not copied to memory due to a receive DMA overrun.

32.5.26.15 *Receive Symbol Errors Register*

Register Name: EMAC_RSE

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
RSE							

- **RSE: Receive Symbol Errors**

An 8-bit register counting the number of frames that had `rx_er` asserted during reception. Receive symbol errors are also counted as an FCS or alignment error if the frame is between 64 and 1518 bytes in length (1536 if bit 8 is set in the network configuration register). If the frame is larger, it is recorded as a jabber error.

32.5.26.16 *Excessive Length Errors Register*

Register Name: EMAC_ELE

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
EXL							

- **EXL: Excessive Length Errors**

An 8-bit register counting the number of frames received exceeding 1518 bytes (1536 if bit 8 set in network configuration register) in length but do not have either a CRC error, an alignment error nor a receive symbol error.

32.5.26.17 Receive Jabbers Register

Register Name: EMAC_RJA

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
RJB							

- **RJB: Receive Jabbers**

An 8-bit register counting the number of frames received exceeding 1518 bytes (1536 if bit 8 set in network configuration register) in length and have either a CRC error, an alignment error or a receive symbol error.

32.5.26.18 Undersize Frames Register

Register Name: EMAC_USF

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
USF							

- **USF: Undersize frames**

An 8-bit register counting the number of frames received less than 64 bytes in length but do not have either a CRC error, an alignment error or a receive symbol error.

32.5.26.19 SQE Test Errors Register

Register Name: EMAC_STE

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**
SQER							

- **SQER: SQE test errors**

An 8-bit register counting the number of frames where `col` was not asserted within 96 bit times (an interframe gap) of `tx_en` being deasserted in half duplex mode.

32.5.26.20 Received Length Field Mismatch Register

Register Name: EMAC_RLE

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**
RLFM							

- **RLFM: Receive Length Field Mismatch**

An 8-bit register counting the number of frames received that have a measured length shorter than that extracted from its length field. Checking is enabled through bit 16 of the network configuration register. Frames containing a type ID in bytes 13 and 14 (i.e., length/type ID $\geq$ 0x0600) are not counted as length field errors, neither are excessive length frames.

33. Controller Area Network (CAN)

33.1 Description

The CAN controller provides all the features required to implement the serial communication protocol CAN defined by Robert Bosch GmbH, the CAN specification as referred to by ISO/11898A (2.0 Part A and 2.0 Part B) for high speeds and ISO/11519-2 for low speeds. The CAN Controller is able to handle all types of frames (Data, Remote, Error and Overload) and achieves a bitrate of 1 Mbit/sec.

CAN controller accesses are made through configuration registers. 16 independent message objects (mailboxes) are implemented.

Any mailbox can be programmed as a reception buffer block (even non-consecutive buffers). For the reception of defined messages, one or several message objects can be masked without participating in the buffer feature. An interrupt is generated when the buffer is full. According to the mailbox configuration, the first message received can be locked in the CAN controller registers until the application acknowledges it, or this message can be discarded by new received messages.

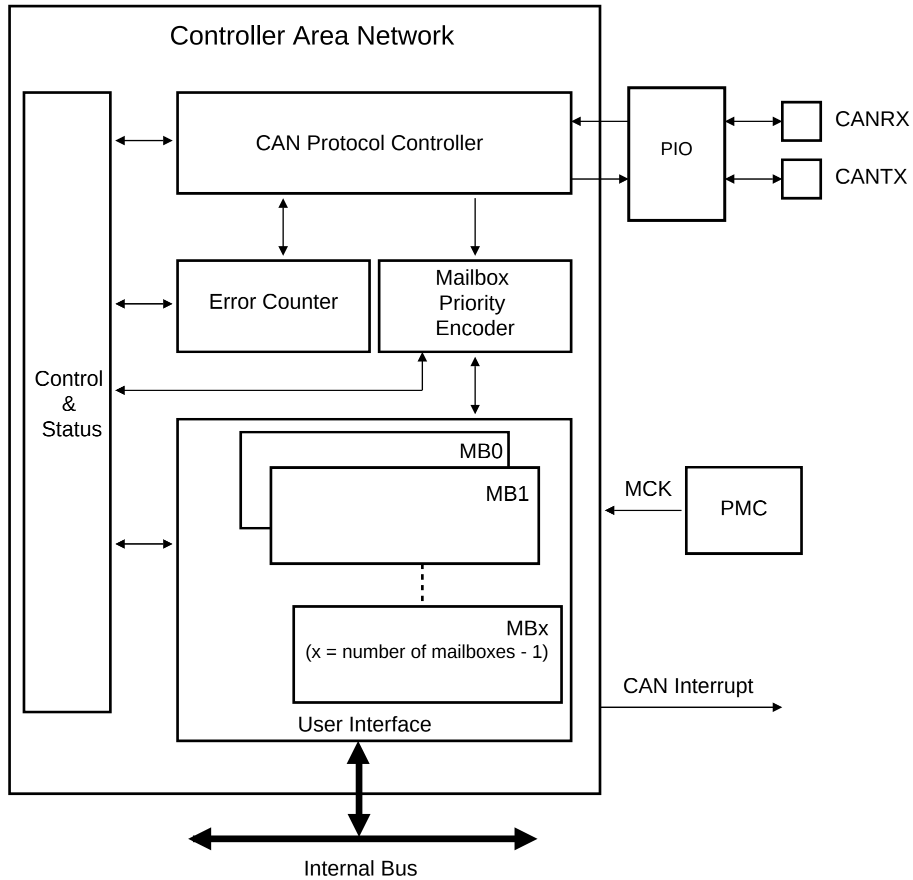
Any mailbox can be programmed for transmission. Several transmission mailboxes can be enabled in the same time. A priority can be defined for each mailbox independently.

An internal 16-bit timer is used to stamp each received and sent message. This timer starts counting as soon as the CAN controller is enabled. This counter can be reset by the application or automatically after a reception in the last mailbox in Time Triggered Mode.

The CAN controller offers optimized features to support the Time Triggered Communication (TTC) protocol.

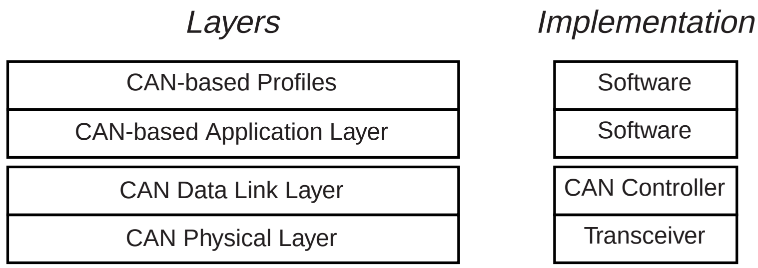
33.2 Block Diagram

Figure 33-1. CAN Block Diagram



33.3 Application Block Diagram

Figure 33-2. Application Block Diagram



33.4 I/O Lines Description

Table 33-1. I/O Lines Description

Name	Description	Type
CANRX	CAN Receive Serial Data	Input
CANTX	CAN Transmit Serial Data	Output

33.5 Product Dependencies

33.5.1 I/O Lines

The pins used for interfacing the CAN may be multiplexed with the PIO lines. The programmer must first program the PIO controller to assign the desired CAN pins to their peripheral function. If I/O lines of the CAN are not used by the application, they can be used for other purposes by the PIO Controller.

33.5.2 Power Management

The programmer must first enable the CAN clock in the Power Management Controller (PMC) before using the CAN.

A Low-power Mode is defined for the CAN controller: If the application does not require CAN operations, the CAN clock can be stopped when not needed and be restarted later. Before stopping the clock, the CAN Controller must be in Low-power Mode to complete the current transfer. After restarting the clock, the application must disable the Low-power Mode of the CAN controller.

33.5.3 Interrupt

The CAN interrupt line is connected on one of the internal sources of the Advanced Interrupt Controller. Using the CAN interrupt requires the AIC to be programmed first. Note that it is not recommended to use the CAN interrupt line in edge-sensitive mode.

33.6 CAN Controller Features

33.6.1 CAN Protocol Overview

The Controller Area Network (CAN) is a multi-master serial communication protocol that efficiently supports real-time control with a very high level of security with bit rates up to 1 Mbit/s.

The CAN protocol supports four different frame types:

- Data frames: They carry data from a transmitter node to the receiver nodes. The overall maximum data frame length is 108 bits for a standard frame and 128 bits for an extended frame.
- Remote frames: A destination node can request data from the source by sending a remote frame with an identifier that matches the identifier of the required data frame. The appropriate data source node then sends a data frame as a response to this node request.
- Error frames: An error frame is generated by any node that detects a bus error.
- Overload frames: They provide an extra delay between the preceding and the successive data frames or remote frames.

The Atmel CAN controller provides the CPU with full functionality of the CAN protocol V2.0 Part A and V2.0 Part B. It minimizes the CPU load in communication overhead. The Data Link Layer and part of the physical layer are automatically handled by the CAN controller itself.

The CPU reads or writes data or messages via the CAN controller mailboxes. An identifier is assigned to each mailbox. The CAN controller encapsulates or decodes data messages to build or to decode bus data frames. Remote frames, error frames and overload frames are automatically handled by the CAN controller under supervision of the software application.

33.6.2 Mailbox Organization

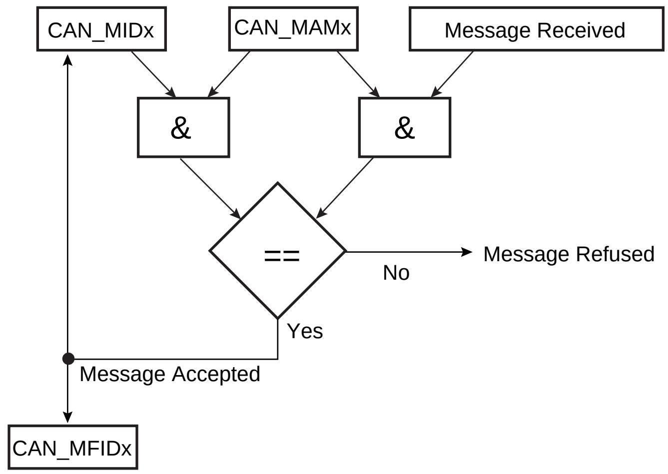
The CAN module has 16 buffers, also called channels or mailboxes. An identifier that corresponds to the CAN identifier is defined for each active mailbox. Message identifiers can match the standard frame identifier or the extended frame identifier. This identifier is defined for the first time during the CAN initialization, but can be dynamically reconfigured later so that the mailbox can handle a new message family. Several mailboxes can be configured with the same ID.

Each mailbox can be configured in receive or in transmit mode independently. The mailbox object type is defined in the MOT field of the CAN_MMRx register.

33.6.2.1 Message Acceptance Procedure

If the MIDE field in the CAN_MIDx register is set, the mailbox can handle the extended format identifier; otherwise, the mailbox handles the standard format identifier. Once a new message is received, its ID is masked with the CAN_MAMx value and compared with the CAN_MIDx value. If accepted, the message ID is copied to the CAN_MIDx register.

Figure 33-3. Message Acceptance Procedure



If a mailbox is dedicated to receiving several messages (a family of messages) with different IDs, the acceptance mask defined in the CAN_MAMx register must mask the variable part of the ID family. Once a message is received, the application must decode the masked bits in the CAN_MIDx. To speed up the decoding, masked bits are grouped in the family ID register (CAN_MFIDx).

For example, if the following message IDs are handled by the same mailbox:

```
ID0 101000100100010010000100 0 11 00b
ID1 101000100100010010000100 0 11 01b
ID2 101000100100010010000100 0 11 10b
ID3 101000100100010010000100 0 11 11b
ID4 101000100100010010000100 1 11 00b
ID5 101000100100010010000100 1 11 01b
ID6 101000100100010010000100 1 11 10b
ID7 101000100100010010000100 1 11 11b
```

The CAN_MIDx and CAN_MAMx of Mailbox x must be initialized to the corresponding values:

```
CAN_MIDx = 001 101000100100010010000100 x 11 xxb
CAN_MAMx = 001 111111111111111111111111 0 11 00b
```

If Mailbox x receives a message with ID6, then CAN_MIDx and CAN_MFIDx are set:

```
CAN_MIDx = 001 101000100100010010000100 1 11 10b
CAN_MFIDx = 00000000000000000000000000000110b
```

If the application associates a handler for each message ID, it may define an array of pointers to functions:

```
void (*pHandler[8])(void);
```

When a message is received, the corresponding handler can be invoked using CAN_MFIDx register and there is no need to check masked bits:

```
unsigned int MFID0_register;
MFID0_register = Get_CAN_MFID0_Register();
// Get_CAN_MFID0_Register() returns the value of the CAN_MFID0 register
pHandler[MFID0_register]();
```

33.6.2.2 Receive Mailbox

When the CAN module receives a message, it looks for the first available mailbox with the lowest number and compares the received message ID with the mailbox ID. If such a mailbox is found, then the message is stored in its data registers. Depending on the configuration, the mailbox is disabled as long as the message has not been acknowledged by the application (Receive only), or, if new messages with the same ID are received, then they overwrite the previous ones (Receive with overwrite).

It is also possible to configure a mailbox in Consumer Mode. In this mode, after each transfer request, a remote frame is automatically sent. The first answer received is stored in the corresponding mailbox data registers.

Several mailboxes can be chained to receive a buffer. They must be configured with the same ID in Receive Mode, except for the last one, which can be configured in Receive with Overwrite Mode. The last mailbox can be used to detect a buffer overflow.

Mailbox Object Type	Description
Receive	The first message received is stored in mailbox data registers. Data remain available until the next transfer request.
Receive with overwrite	The last message received is stored in mailbox data register. The next message always overwrites the previous one. The application has to check whether a new message has not overwritten the current one while reading the data registers.
Consumer	A remote frame is sent by the mailbox. The answer received is stored in mailbox data register. This extends Receive mailbox features. Data remain available until the next transfer request.

33.6.2.3 Transmit Mailbox

When transmitting a message, the message length and data are written to the transmit mailbox with the correct identifier. For each transmit mailbox, a priority is assigned. The controller automatically sends the message with the highest priority first (set with the field PRIOR in CAN_MMRx register).

It is also possible to configure a mailbox in Producer Mode. In this mode, when a remote frame is received, the mailbox data are sent automatically. By enabling this mode, a producer can be done using only one mailbox instead of two: one to detect the remote frame and one to send the answer.

Mailbox Object Type	Description
Transmit	The message stored in the mailbox data registers will try to win the bus arbitration immediately or later according to or not the Time Management Unit configuration (see Section 33.6.3). The application is notified that the message has been sent or aborted.
Producer	The message prepared in the mailbox data registers will be sent after receiving the next remote frame. This extends transmit mailbox features.

33.6.3 Time Management Unit

The CAN Controller integrates a free-running 16-bit internal timer. The counter is driven by the bit clock of the CAN bus line. It is enabled when the CAN controller is enabled (CANEN set in the CAN_MR register). It is automatically cleared in the following cases:

- after a reset
- when the CAN controller is in Low-power Mode is enabled (LPM bit set in the CAN_MR and SLEEP bit set in the CAN_SR)
- after a reset of the CAN controller (CANEN bit in the CAN_MR register)
- in Time-triggered Mode, when a message is accepted by the last mailbox (rising edge of the MRDY signal in the $CAN\_MSR_{last\_mailbox\_number}$ register).

The application can also reset the internal timer by setting TIMRST in the CAN_TCR register. The current value of the internal timer is always accessible by reading the CAN_TIM register.

When the timer rolls-over from FFFFh to 0000h, TOVF (Timer Overflow) signal in the CAN_SR register is set. TOVF bit in the CAN_SR register is cleared by reading the CAN_SR register. Depending on the corresponding interrupt mask in the CAN_IMR register, an interrupt is generated while TOVF is set.

In a CAN network, some CAN devices may have a larger counter. In this case, the application can also decide to freeze the internal counter when the timer reaches FFFFh and to wait for a restart condition from another device. This feature is enabled by setting TIMFRZ in the CAN_MR register. The CAN_TIM register is frozen to the FFFFh value. A clear condition described above restarts the timer. A timer overflow (TOVF) interrupt is triggered.

To monitor the CAN bus activity, the CAN_TIM register is copied to the CAN _TIMESTP register after each start of frame or end of frame and a TSTP interrupt is triggered. If TEOF bit in the CAN_MR register is set, the value is captured at each End Of Frame, else it is captured at each Start Of Frame. Depending on the corresponding mask in the CAN_IMR register, an interrupt is generated while TSTP is set in the CAN_SR. TSTP bit is cleared by reading the CAN_SR register.

The time management unit can operate in one of the two following modes:

- Timestamping mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame
- Time Triggered mode: A mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing TTM field in the CAN_MR register. Time Triggered Mode is enabled by setting TTM field in the CAN_MR register.

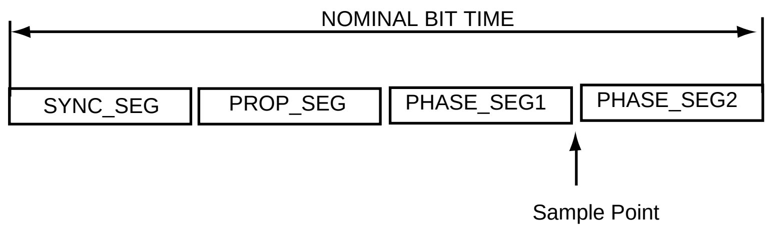
33.6.4 CAN 2.0 Standard Features

33.6.4.1 CAN Bit Timing Configuration

All controllers on a CAN bus must have the same bit rate and bit length. At different clock frequencies of the individual controllers, the bit rate has to be adjusted by the time segments.

The CAN protocol specification partitions the nominal bit time into four different segments:

Figure 33-4. Partition of the CAN Bit Time



TIME QUANTUM:

The TIME QUANTUM (TQ) is a fixed unit of time derived from the MCK period. The total number of TIME QUANTA in a bit time is programmable from 8 to 25.

SYNC SEG: SYNChronization Segment.

This part of the bit time is used to synchronize the various nodes on the bus. An edge is expected to lie within this segment. It is 1 TQ long.

PROP SEG: PROPagation Segment.

This part of the bit time is used to compensate for the physical delay times within the network. It is twice the sum of the signal's propagation time on the bus line, the input comparator delay, and the output driver delay. It is programmable to be 1,2,..., 8 TQ long.

This parameter is defined in the PROPAG field of the "CAN Baudrate Register".

PHASE SEG1, PHASE SEG2: PHASE Segment 1 and 2.

The Phase-Buffer-Segments are used to compensate for edge phase errors. These segments can be lengthened (PHASE SEG1) or shortened (PHASE SEG2) by resynchronization.

Phase Segment 1 is programmable to be 1,2,..., 8 TQ long.

Phase Segment 2 length has to be at least as long as the Information Processing Time (IPT) and may not be more than the length of Phase Segment 1.

These parameters are defined in the PHASE1 and PHASE2 fields of the "CAN Baudrate Register".

INFORMATION PROCESSING TIME:

The Information Processing Time (IPT) is the time required for the logic to determine the bit level of a sampled bit. The IPT begins at the sample point, is measured in TQ and **is fixed at 2 TQ for the Atmel CAN**. Since Phase Segment 2 also begins at the sample point and is the last segment in the bit time, PHASE SEG2 shall not be less than the IPT.

SAMPLE POINT:

The SAMPLE POINT is the point in time at which the bus level is read and interpreted as the value of that respective bit. Its location is at the end of PHASE_SEG1.

SJW: ReSynchronization Jump Width.

The ReSynchronization Jump Width defines the limit to the amount of lengthening or shortening of the Phase Segments.

SJW is programmable to be the minimum of PHASE SEG1 and 4 TQ.

If the SMP field in the CAN_BR register is set, then the incoming bit stream is sampled three times with a period of half a CAN clock period, centered on sample point.

In the CAN controller, the length of a bit on the CAN bus is determined by the parameters (BRP, PROPAG, PHASE1 and PHASE2).

$$t_{BIT} = t_{CSC} + t_{PRS} + t_{PHS1} + t_{PHS2}$$

The time quantum is calculated as follows:

$$t_{CSC} = (BRP + 1) / MCK$$

Note: The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

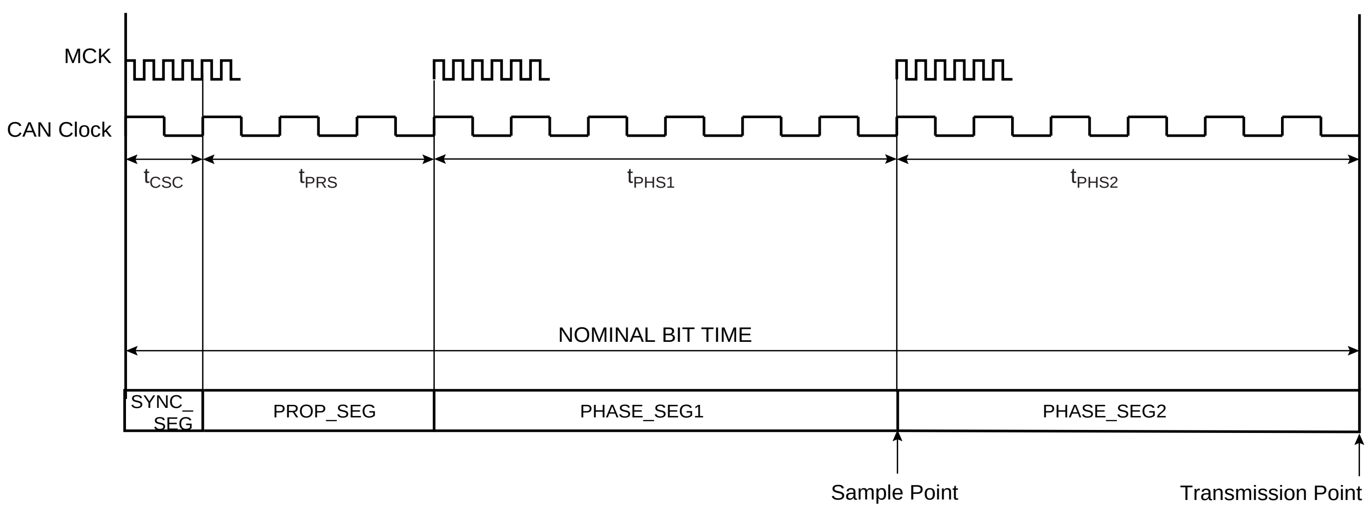
$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

To compensate for phase shifts between clock oscillators of different controllers on the bus, the CAN controller must resynchronize on any relevant signal edge of the current transmission. The resynchronization shortens or lengthens the bit time so that the position of the sample point is shifted with regard to the detected edge. The resynchronization jump width (SJW) defines the maximum of time by which a bit period may be shortened or lengthened by resynchronization.

$$t_{SJW} = t_{CSC} \times (SJW + 1)$$

Figure 33-5. CAN Bit Timing



Example of bit timing determination for CAN baudrate of 500 Kbit/s:

```
MCK = 48MHz
CAN baudrate= 500kbit/s => bit time= 2us
```

```
Delay of the bus driver: 50 ns
Delay of the receiver: 30ns
Delay of the bus line (20m): 110ns
```

```
The total number of time quanta in a bit time must be comprised between 8
and 25. If we fix the bit time to 16 time quanta:
Tcsc = 1 time quanta = bit time / 16 = 125 ns
=> BRP = (Tcsc x MCK) - 1 = 5
```

```
The propagation segment time is equal to twice the sum of the signal's
propagation time on the bus line, the receiver delay and the output driver
delay:
Tprs = 2 * (50+30+110) ns = 380 ns = 3 Tcsc
=> PROPAG = Tprs/Tcsc - 1 = 2
The remaining time for the two phase segments is:
Tphs1 + Tphs2 = bit time - Tcsc - Tprs = (16 - 1 - 3)Tcsc
Tphs1 + Tphs2 = 12 Tcsc
```

```
Because this number is even, we choose Tphs2 = Tphs1 (else we would choose
Tphs2 = Tphs1 + Tcsc)
Tphs1 = Tphs2 = (12/2) Tcsc = 6 Tcsc
=> PHASE1 = PHASE2 = Tphs1/Tcsc - 1 = 5
```

```
The resynchronization jump width must be comprised between 1 Tcsc and the
minimum of 4 Tcsc and Tphs1. We choose its maximum value:
Tsjw = Min(4 Tcsc,Tphs1) = 4 Tcsc
=> SJW = Tsjw/Tcsc - 1 = 3
```

```
Finally: CAN_BR = 0x00053255
```

33.6.4.1.1 CAN Bus Synchronization

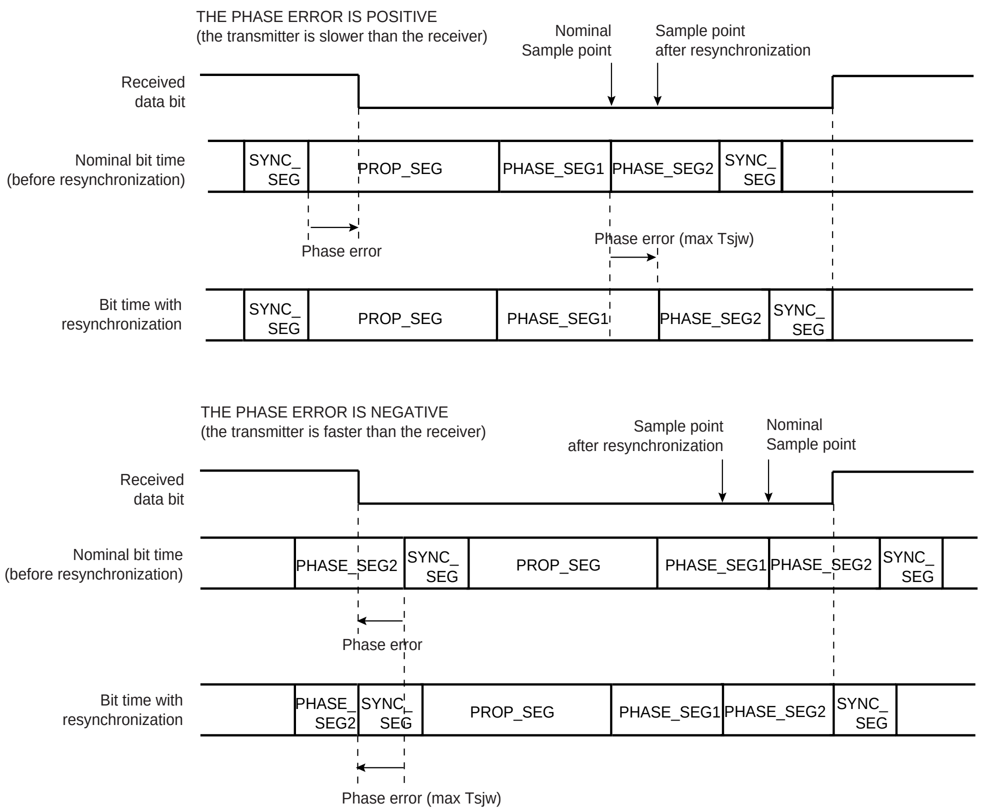
Two types of synchronization are distinguished: "hard synchronization" at the start of a frame and "resynchronization" inside a frame. After a hard synchronization, the bit time is restarted with the end of the SYNC_SEG segment, regardless of the phase error. Resynchronization causes a reduction or increase in the bit time so that the position of the sample point is shifted with respect to the detected edge.

The effect of resynchronization is the same as that of hard synchronization when the magnitude of the phase error of the edge causing the resynchronization is less than or equal to the programmed value of the resynchronization jump width (t_{SJW}).

When the magnitude of the phase error is larger than the resynchronization jump width and

- the phase error is positive, then PHASE_SEG1 is lengthened by an amount equal to the resynchronization jump width.
- the phase error is negative, then PHASE_SEG2 is shortened by an amount equal to the resynchronization jump width.

Figure 33-6. CAN Resynchronization



33.6.4.1.1 Autobaud Mode

The autobaud feature is enabled by setting the ABM field in the CAN_MR register. In this mode, the CAN controller is only listening to the line without acknowledging the received messages. It can not send any message. The errors flags are updated. The bit timing can be adjusted until no error occurs (good configuration found). In this mode, the error counters are frozen. To go back to the standard mode, the ABM bit must be cleared in the CAN_MR register.

33.6.4.2 Error Detection

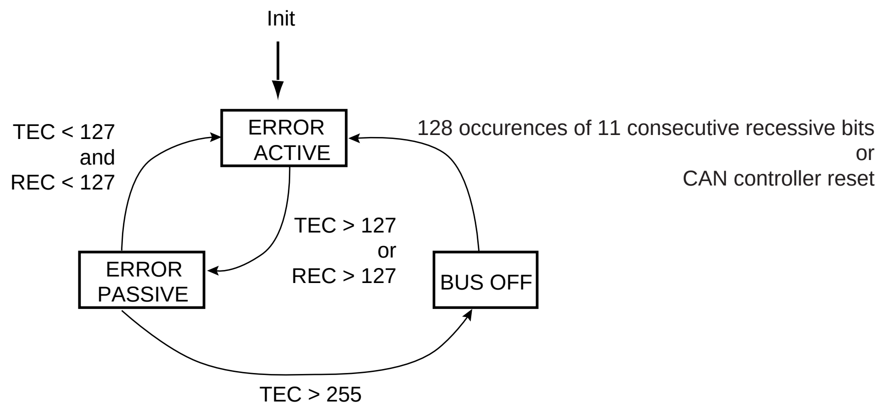
There are five different error types that are not mutually exclusive. Each error concerns only specific fields of the CAN data frame (refer to the Bosch CAN specification for their correspondence):

- CRC error (CERR bit in the CAN_SR register): With the CRC, the transmitter calculates a checksum for the CRC bit sequence from the Start of Frame bit until the end of the Data Field. This CRC sequence is transmitted in the CRC field of the Data or Remote Frame.
- Bit-stuffing error (SERR bit in the CAN_SR register): If a node detects a sixth consecutive equal bit level during the bit-stuffing area of a frame, it generates an Error Frame starting with the next bit-time.
- Bit error (BERR bit in CAN_SR register): A bit error occurs if a transmitter sends a dominant bit but detects a recessive bit on the bus line, or if it sends a recessive bit but detects a dominant bit on the bus line. An error frame is generated and starts with the next bit time.
- Form Error (FERR bit in the CAN_SR register): If a transmitter detects a dominant bit in one of the fix-formatted segments CRC Delimiter, ACK Delimiter or End of Frame, a form error has occurred and an error frame is generated.
- Acknowledgment error (AERR bit in the CAN_SR register): The transmitter checks the Acknowledge Slot, which is transmitted by the transmitting node as a recessive bit, contains a dominant bit. If this is the case, at least one other node has received the frame correctly. If not, an Acknowledge Error has occurred and the transmitter will start in the next bit-time an Error Frame transmission.

33.6.4.2.1 Fault Confinement

To distinguish between temporary and permanent failures, every CAN controller has two error counters: REC (Receive Error Counter) and TEC (Transmit Error Counter). The counters are incremented upon detected errors and respectively are decremented upon correct transmissions or receptions. Depending on the counter values, the state of the node changes: the initial state of the CAN controller is Error Active, meaning that the controller can send Error Active flags. The controller changes to the Error Passive state if there is an accumulation of errors. If the CAN controller fails or if there is an extreme accumulation of errors, there is a state transition to Bus Off.

Figure 33-7. Line Error Mode



An error active unit takes part in bus communication and sends an active error frame when the CAN controller detects an error.

An error passive unit cannot send an active error frame. It takes part in bus communication, but when an error is detected, a passive error frame is sent. Also, after a transmission, an error passive unit waits before initiating further transmission.

A bus off unit is not allowed to have any influence on the bus.

For fault confinement, two errors counters (TEC and REC) are implemented. These counters are accessible via the CAN_ECR register. The state of the CAN controller is automatically updated according to these counter values. If the CAN controller is in Error Active state, then the ERRA bit is set in the CAN_SR register. The corresponding interrupt is pending while the interrupt is not masked in the CAN_IMR register. If the CAN controller is in Error Passive Mode, then the ERRP bit is set in the CAN_SR register and an interrupt remains pending while the ERRP bit is set in the CAN_IMR register. If the CAN is in Bus-off Mode, then the BOFF bit is set in the CAN_SR register. As for ERRP and ERRA, an interrupt is pending while the BOFF bit is set in the CAN_IMR register.

When one of the error counters values exceeds 96, an increased error rate is indicated to the controller through the WARN bit in CAN_SR register, but the node remains error active. The corresponding interrupt is pending while the interrupt is set in the CAN_IMR register.

Refer to the Bosch CAN specification v2.0 for details on fault confinement.

33.6.4.3 Overload

The overload frame is provided to request a delay of the next data or remote frame by the receiver node (“Request overload frame”) or to signal certain error conditions (“Reactive overload frame”) related to the intermission field respectively.

Reactive overload frames are transmitted after detection of the following error conditions:

- Detection of a dominant bit during the first two bits of the intermission field
- Detection of a dominant bit in the last bit of EOF by a receiver, or detection of a dominant bit by a receiver or a transmitter at the last bit of an error or overload frame delimiter

The CAN controller can generate a request overload frame automatically after each message sent to one of the CAN controller mailboxes. This feature is enabled by setting the OVL bit in the CAN_MR register.

Reactive overload frames are automatically handled by the CAN controller even if the OVL bit in the CAN_MR register is not set. An overload flag is generated in the same way as an error flag, but error counters do not increment.

33.6.5 Low-power Mode

In Low-power Mode, the CAN controller cannot send or receive messages. All mailboxes are inactive.

In Low-power Mode, the SLEEP signal in the CAN_SR register is set; otherwise, the WAKEUP signal in the CAN_SR register is set. These two fields are exclusive except after a CAN controller reset (WAKEUP and SLEEP are stuck at 0 after a reset). After power-up reset, the Low-power Mode is disabled and the WAKEUP bit is set in the CAN_SR register only after detection of 11 consecutive recessive bits on the bus.

33.6.5.1 Enabling Low-power Mode

A software application can enable Low-power Mode by setting the LPM bit in the CAN_MR global register. The CAN controller enters Low-power Mode once all pending transmit messages are sent.

When the CAN controller enters Low-power Mode, the SLEEP signal in the CAN_SR register is set. Depending on the corresponding mask in the CAN_IMR register, an interrupt is generated while SLEEP is set.

The SLEEP signal in the CAN_SR register is automatically cleared once WAKEUP is set. The WAKEUP signal is automatically cleared once SLEEP is set.

Reception is disabled while the SLEEP signal is set to one in the CAN_SR register. It is important to note that those messages with higher priority than the last message transmitted can be received between the LPM command and entry in Low-power Mode.

Once in Low-power Mode, the CAN controller clock can be switched off by programming the chip's Power Management Controller (PMC). The CAN controller drains only the static current.

Error counters are disabled while the SLEEP signal is set to one.

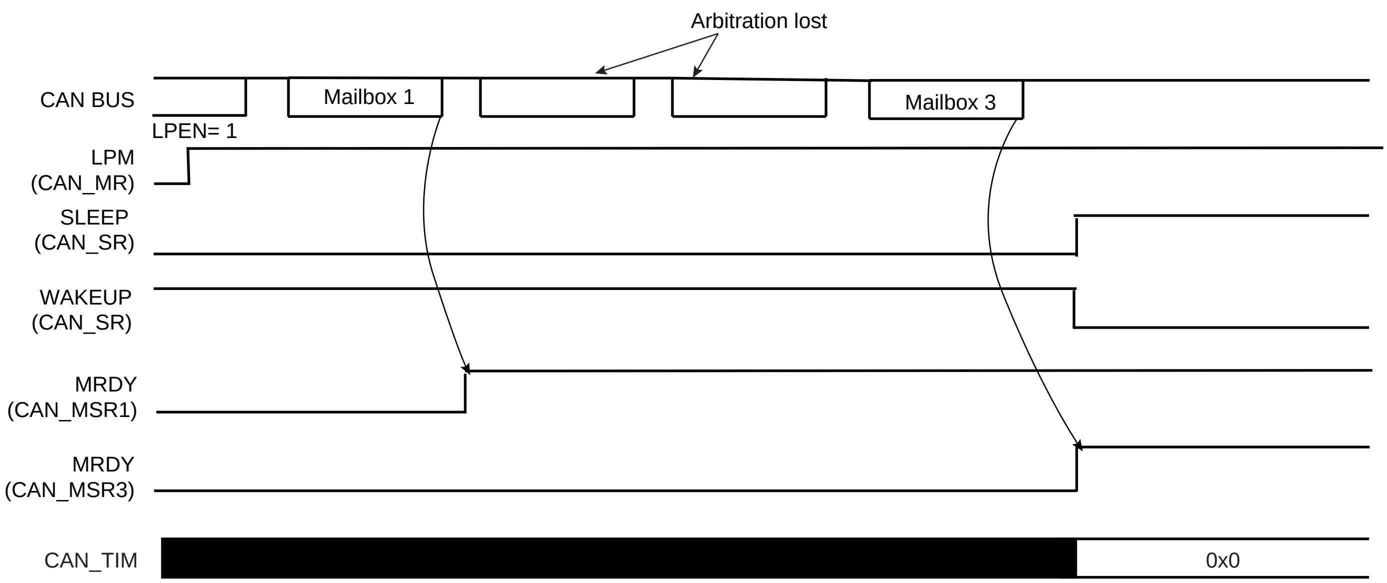
Thus, to enter Low-power Mode, the software application must:

– Set LPM field in the CAN_MR register

– Wait for SLEEP signal rising

Now the CAN Controller clock can be disabled. This is done by programming the Power Management Controller (PMC).

Figure 33-8. Enabling Low-power Mode



33.6.5.2 Disabling Low-power Mode

The CAN controller can be awake after detecting a CAN bus activity. Bus activity detection is done by an external module that may be embedded in the chip. When it is notified of a CAN bus activity, the software application disables Low-power Mode by programming the CAN controller.

To disable Low-power Mode, the software application must:

– Enable the CAN Controller clock. This is done by programming the Power Management Controller (PMC).

– Clear the LPM field in the CAN_MR register

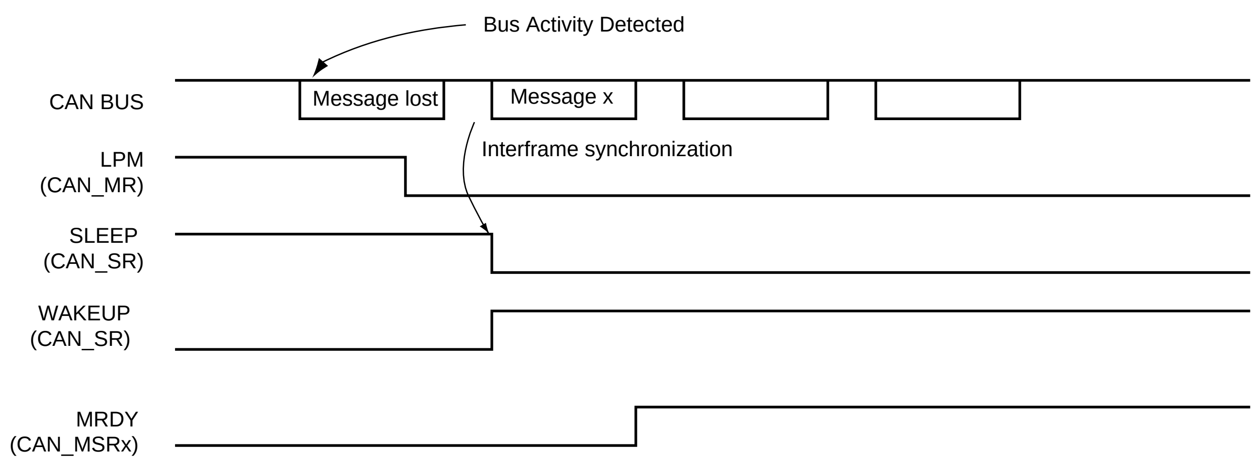
The CAN controller synchronizes itself with the bus activity by checking for eleven consecutive “recessive” bits. Once synchronized, the WAKEUP signal in the CAN_SR register is set.

Depending on the corresponding mask in the CAN_IMR register, an interrupt is generated while WAKEUP is set. The SLEEP signal in the CAN_SR register is automatically cleared once WAKEUP is set. WAKEUP signal is automatically cleared once SLEEP is set.

If no message is being sent on the bus, then the CAN controller is able to send a message eleven bit times after disabling Low-power Mode.

If there is bus activity when Low-power mode is disabled, the CAN controller is synchronized with the bus activity in the next interframe. The previous message is lost (see Figure 33-9).

Figure 33-9. Disabling Low-power Mode



33.7 Functional Description

33.7.1 CAN Controller Initialization

After power-up reset, the CAN controller is disabled. The CAN controller clock must be activated by the Power Management Controller (PMC) and the CAN controller interrupt line must be enabled by the interrupt controller (AIC).

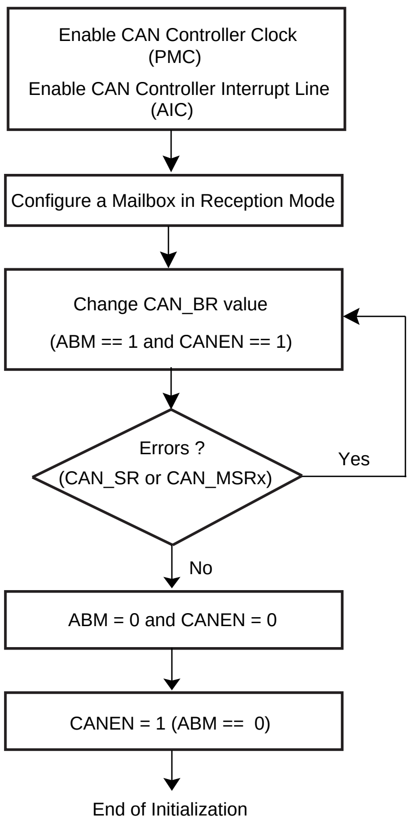
The CAN controller must be initialized with the CAN network parameters. The CAN_BR register defines the sampling point in the bit time period. CAN_BR must be set before the CAN controller is enabled by setting the CANEN field in the CAN_MR register.

The CAN controller is enabled by setting the CANEN flag in the CAN_MR register. At this stage, the internal CAN controller state machine is reset, error counters are reset to 0, error flags are reset to 0.

Once the CAN controller is enabled, bus synchronization is done automatically by scanning eleven recessive bits. The WAKEUP bit in the CAN_SR register is automatically set to 1 when the CAN controller is synchronized (WAKEUP and SLEEP are stuck at 0 after a reset).

The CAN controller can start listening to the network in Autobaud Mode. In this case, the error counters are locked and a mailbox may be configured in Receive Mode. By scanning error flags, the CAN_BR register values synchronized with the network. Once no error has been detected, the application disables the Autobaud Mode, clearing the ABM field in the CAN_MR register.

Figure 33-10. Possible Initialization Procedure



33.7.2 CAN Controller Interrupt Handling

There are two different types of interrupts. One type of interrupt is a message-object related interrupt, the other is a system interrupt that handles errors or system-related interrupt sources.

All interrupt sources can be masked by writing the corresponding field in the CAN_IDR register. They can be unmasked by writing to the CAN_IER register. After a power-up reset, all interrupt sources are disabled (masked). The current mask status can be checked by reading the CAN_IMR register.

The CAN_SR register gives all interrupt source states.

The following events may initiate one of the two interrupts:

- Message object interrupt
  - Data registers in the mailbox object are available to the application. In Receive Mode, a new message was received. In Transmit Mode, a message was transmitted successfully.
  - A sent transmission was aborted.
- System interrupts
  - Bus-off interrupt: The CAN module enters the bus-off state.
  - Error-passive interrupt: The CAN module enters Error Passive Mode.

– Error-active Mode: The CAN module is neither in Error Passive Mode nor in Bus-off mode.
– Warn Limit interrupt: The CAN module is in Error-active Mode, but at least one of its error counter value exceeds 96.
– Wake-up interrupt: This interrupt is generated after a wake-up and a bus synchronization.
– Sleep interrupt: This interrupt is generated after a Low-power Mode enable once all pending messages in transmission have been sent.
– Internal timer counter overflow interrupt: This interrupt is generated when the internal timer rolls over.
– Timestamp interrupt: This interrupt is generated after the reception or the transmission of a start of frame or an end of frame. The value of the internal counter is copied in the CAN_TIMESTP register.

All interrupts are cleared by clearing the interrupt source except for the internal timer counter overflow interrupt and the timestamp interrupt. These interrupts are cleared by reading the CAN_SR register.

33.7.3 CAN Controller Message Handling

33.7.3.1 Receive Handling

Two modes are available to configure a mailbox to receive messages. In ***Receive Mode***, the first message received is stored in the mailbox data register. In ***Receive with Overwrite Mode***, the last message received is stored in the mailbox.

33.7.3.1.1 Simple Receive Mailbox

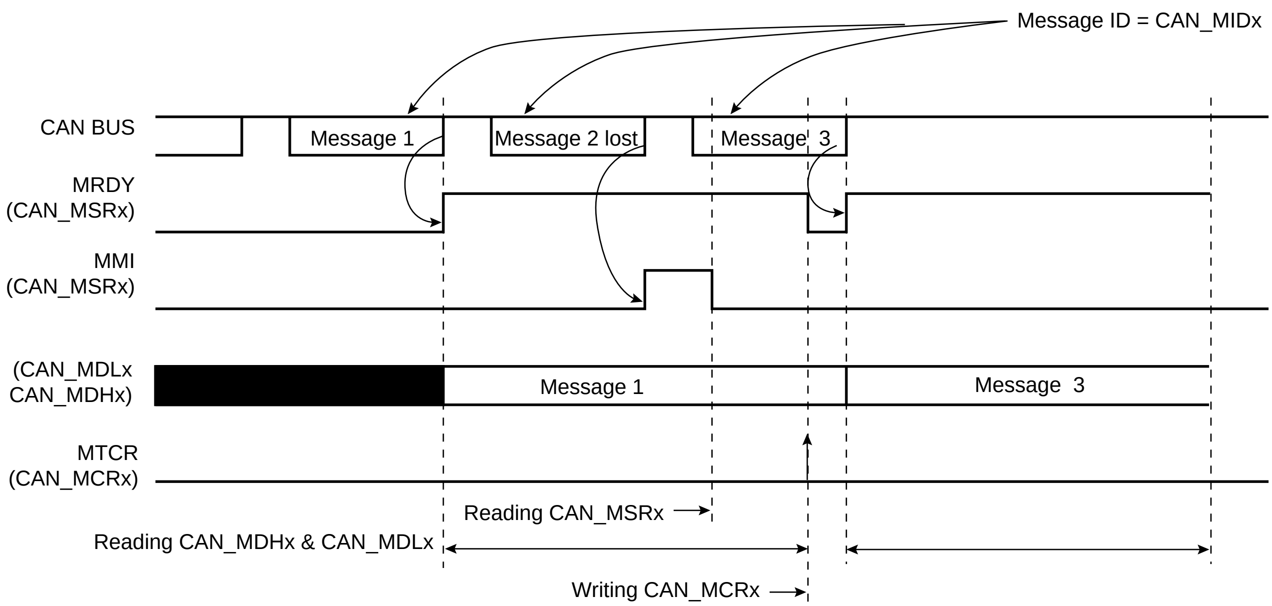
A mailbox is in Receive Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance Mask must be set before the Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN_MSR register is automatically cleared until the first message is received. When the first message has been accepted by the mailbox, the MRDY flag is set. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN_IMR global register.

Message data are stored in the mailbox data register until the software application notifies that data processing has ended. This is done by asking for a new transfer command, setting the MTCR flag in the CAN_MCRx register. This automatically clears the MRDY signal.

The MMI flag in the CAN_MSRx register notifies the software that a message has been lost by the mailbox. This flag is set when messages are received while MRDY is set in the CAN_MSRx register. This flag is cleared by reading the CAN_MSRs register. A receive mailbox prevents from overwriting the first message by new ones while MRDY flag is set in the CAN_MSRx register. See Figure 33-11.

Figure 33-11. Receive Mailbox



Note: In the case of ARM architecture, CAN_MSRx, CAN_MDLx, CAN_MDHx can be read using an optimized ldm assembler instruction.

33.7.3.1.1 Receive with Overwrite Mailbox

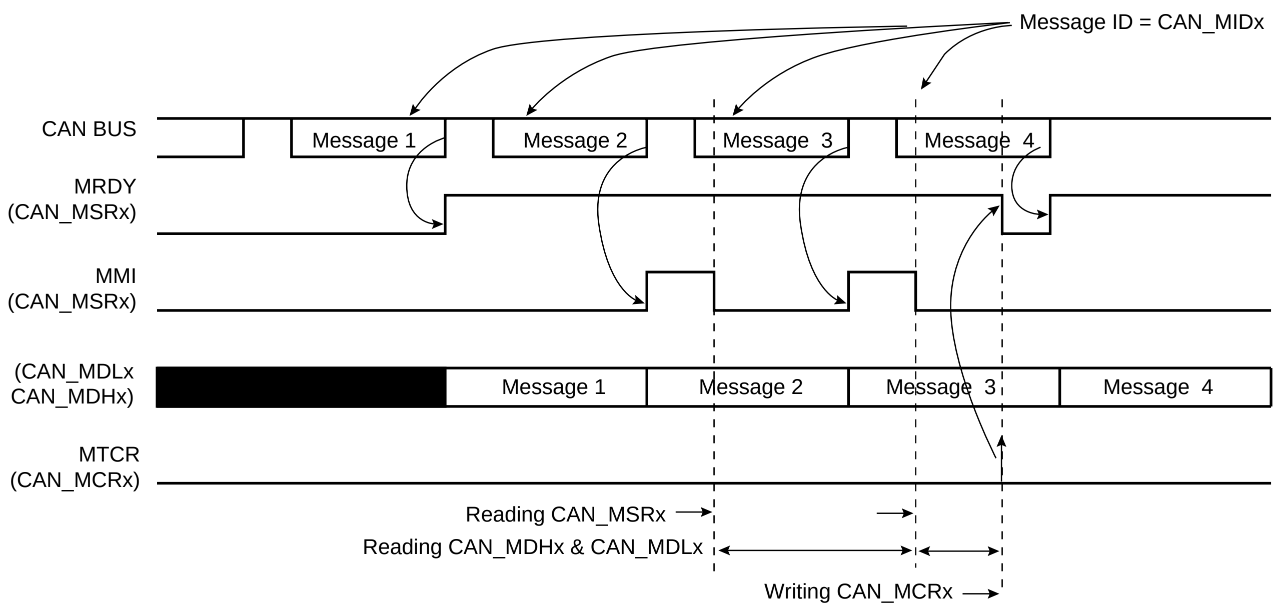
A mailbox is in Receive with Overwrite Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Receive Mode is enabled, the MRDY flag in the CAN_MSR register is automatically cleared until the first message is received. When the first message has been accepted by the mailbox, the MRDY flag is set. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt is masked depending on the mailbox flag in the CAN_IMR global register.

If a new message is received while the MRDY flag is set, this new message is stored in the mailbox data register, overwriting the previous message. The MMI flag in the CAN_MSRx register notifies the software that a message has been dropped by the mailbox. This flag is cleared when reading the CAN_MSRx register.

The CAN controller may store a new message in the CAN data registers while the application reads them. To check that CAN_MDHx and CAN_MDLx do not belong to different messages, the application must check the MMI field in the CAN_MSRx register before and after reading CAN_MDHx and CAN_MDLx. If the MMI flag is set again after the data registers have been read, the software application has to re-read CAN_MDHx and CAN_MDLx (see Figure 33-12).

Figure 33-12. Receive with Overwrite Mailbox

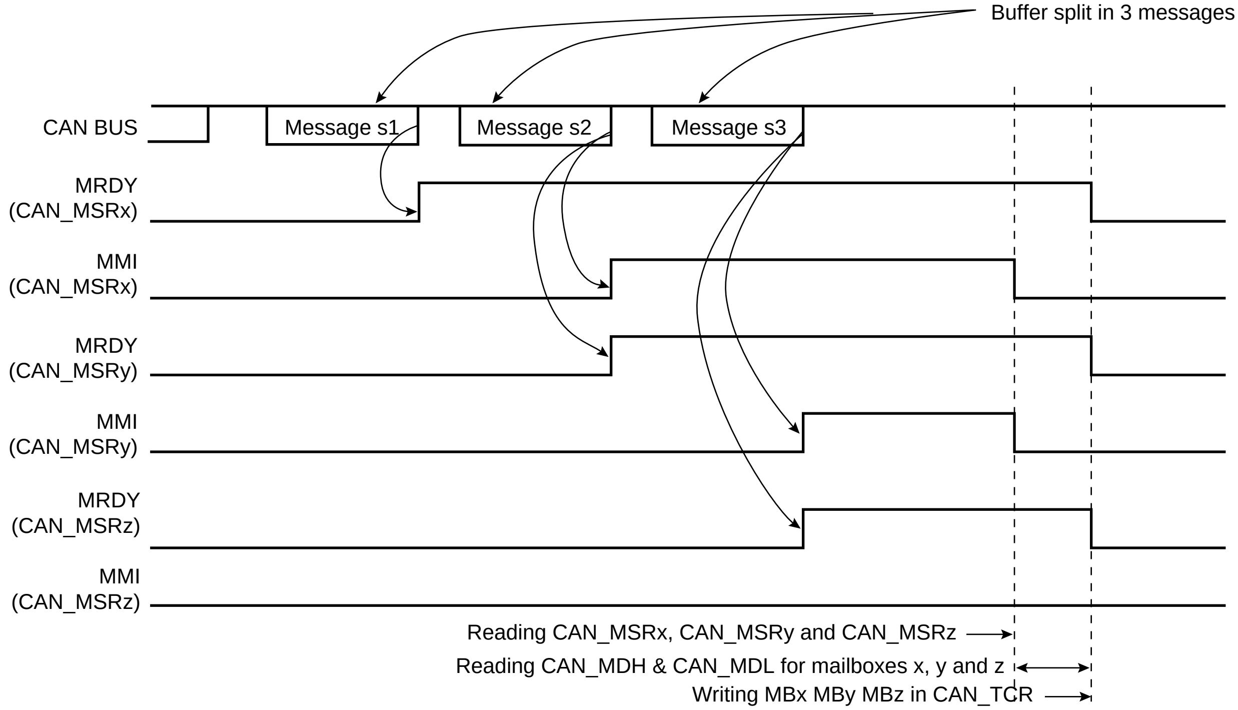


33.7.3.1.1 Chaining Mailboxes

Several mailboxes may be used to receive a buffer split into several messages with the same ID. In this case, the mailbox with the lowest number is serviced first. In the receive and receive with overwrite modes, the field PRIOR in the CAN_MMRx register has no effect. If Mailbox 0 and Mailbox 5 accept messages with the same ID, the first message is received by Mailbox 0 and the second message is received by Mailbox 5. Mailbox 0 must be configured in Receive Mode (i.e., the first message received is considered) and Mailbox 5 must be configured in Receive with Overwrite Mode. Mailbox 0 cannot be configured in Receive with Overwrite Mode; otherwise, all messages are accepted by this mailbox and Mailbox 5 is never serviced.

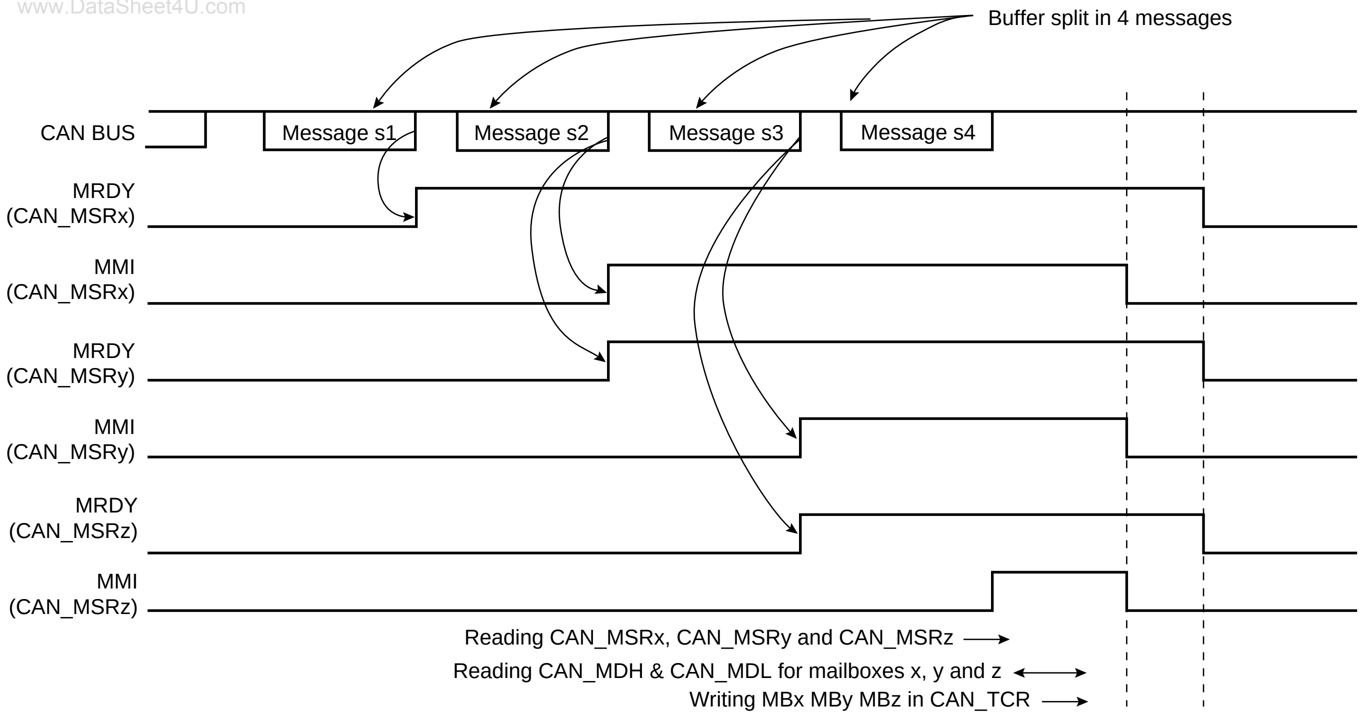
If several mailboxes are chained to receive a buffer split into several messages, all mailboxes except the last one (with the highest number) must be configured in Receive Mode. The first message received is handled by the first mailbox, the second one is refused by the first mailbox and accepted by the second mailbox, the last message is accepted by the last mailbox and refused by previous ones (see Figure 33-13).

Figure 33-13. Chaining Three Mailboxes to Receive a Buffer Split into Three Messages



If the number of mailboxes is not sufficient (the MMI flag of the last mailbox raises), the user must read each data received on the last mailbox in order to retrieve all the messages of the buffer split (see Figure 33-14).

Figure 33-14. Chaining Three Mailboxes to Receive a Buffer Split into Four Messages



33.7.3.2 Transmission Handling

A mailbox is in Transmit Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance mask must be set before Receive Mode is enabled.

After Transmit Mode is enabled, the MRDY flag in the CAN_MSR register is automatically set until the first command is sent. When the MRDY flag is set, the software application can prepare a message to be sent by writing to the CAN_MDx registers. The message is sent once the software asks for a transfer command setting the MTCR bit and the message data length in the CAN_MCRx register.

The MRDY flag remains at zero as long as the message has not been sent or aborted. It is important to note that no access to the mailbox data register is allowed while the MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked depending on the mailbox flag in the CAN_IMR global register.

It is also possible to send a remote frame setting the MRTR bit instead of setting the MDLC field. The answer to the remote frame is handled by another reception mailbox. In this case, the device acts as a consumer but with the help of two mailboxes. It is possible to handle the remote frame emission and the answer reception using only one mailbox configured in Consumer Mode. Refer to the section "Remote Frame Handling" on page 680.

Several messages can try to win the bus arbitration in the same time. The message with the highest priority is sent first. Several transfer request commands can be generated at the same time by setting MBx bits in the CAN_TCR register. The priority is set in the PRIOR field of the CAN_MMRx register. Priority 0 is the highest priority, priority 15 is the lowest priority. Thus it is possible to use a part of the message ID to set the PRIOR field. If two mailboxes have the same priority, the message of the mailbox with the lowest number is sent first. Thus if mailbox 0 and mailbox 5 have the same priority and have a message to send at the same time, then the message of the mailbox 0 is sent first.

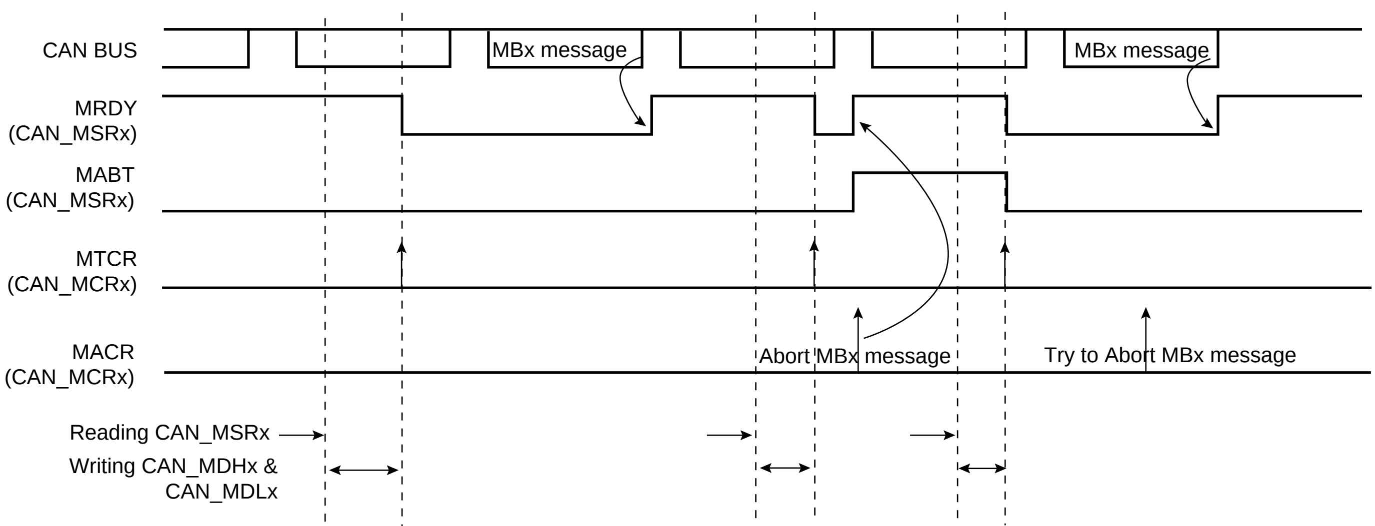
Setting the MACR bit in the CAN_MCRx register aborts the transmission. Transmission for several mailboxes can be aborted by writing MBx fields in the CAN_MACR register. If the message is being sent when the abort command is set, then the application is notified by the MRDY bit set and not the MABT in the CAN_MSRx register. Otherwise, if the message has not been sent, then the MRDY and the MABT are set in the CAN_MSR register.

When the bus arbitration is lost by a mailbox message, the CAN controller tries to win the next bus arbitration with the same message if this one still has the highest priority. Messages to be sent are re-tried automatically until they win the bus arbitration. This feature can be disabled by setting the bit DRPT in the CAN_MR register. In this case if the message was not sent the first time it was transmitted to the CAN transceiver, it is automatically aborted. The MABT flag is set in the CAN_MSRx register until the next transfer command.

Figure 33-15 shows three MBx message attempts being made (MRDY of MBx set to 0).

The first MBx message is sent, the second is aborted and the last one is trying to be aborted but too late because it has already been transmitted to the CAN transceiver.

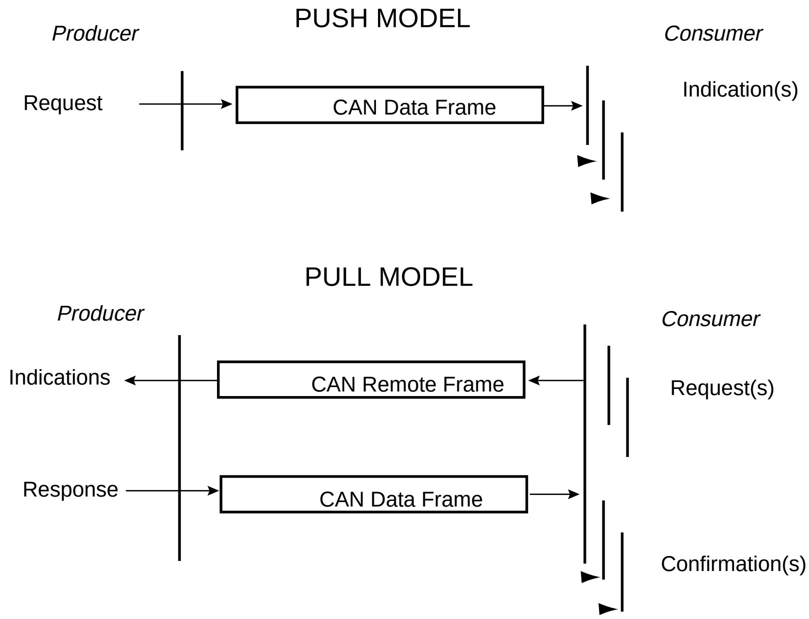
Figure 33-15. Transmitting Messages



33.7.3.3 Remote Frame Handling

Producer/consumer model is an efficient means of handling broadcasted messages. The push model allows a producer to broadcast messages; the pull model allows a customer to ask for messages.

Figure 33-16. Producer / Consumer Model



In Pull Mode, a consumer transmits a remote frame to the producer. When the producer receives a remote frame, it sends the answer accepted by one or many consumers. Using transmit and receive mailboxes, a consumer must dedicate two mailboxes, one in Transmit Mode to send remote frames, and at least one in Receive Mode to capture the producer's answer. The same structure is applicable to a producer: one reception mailbox is required to get the remote frame and one transmit mailbox to answer.

Mailboxes can be configured in Producer or Consumer Mode. A lonely mailbox can handle the remote frame and the answer. With 16 mailboxes, the CAN controller can handle 16 independent producers/consumers.

33.7.3.3.1 Producer Configuration

A mailbox is in Producer Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

After Producer Mode is enabled, the MRDY flag in the CAN_MSR register is automatically set until the first transfer command. The software application prepares data to be sent by writing to the CAN_MDHx and the CAN_MDLx registers, then by setting the MTCR bit in the CAN_MCRx register. Data is sent after the reception of a remote frame as soon as it wins the bus arbitration.

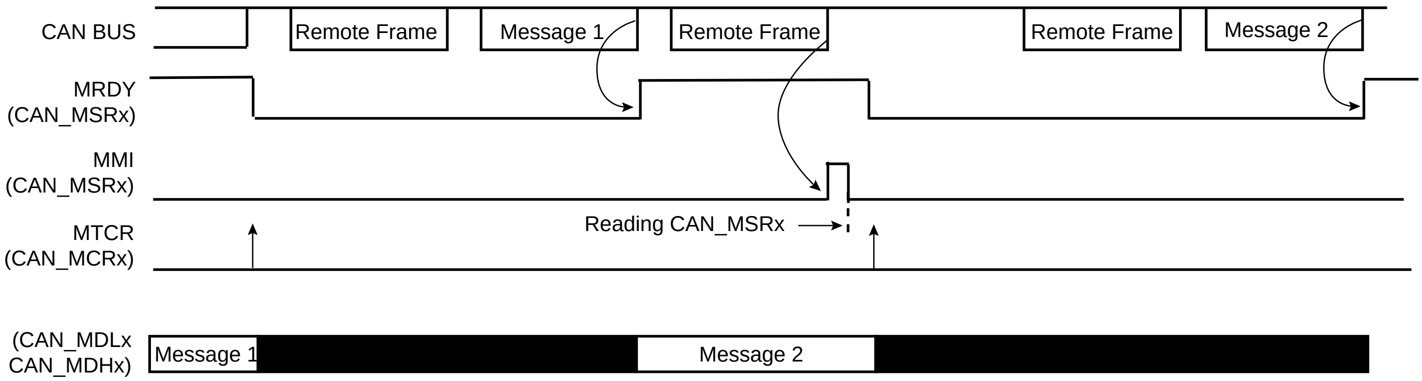
The MRDY flag remains at zero as long as the message has not been sent or aborted. No access to the mailbox data register can be done while MRDY flag is cleared. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN_IMR global register.

If a remote frame is received while no data are ready to be sent (signal MRDY set in the CAN_MSRx register), then the MMI signal is set in the CAN_MSRx register. This bit is cleared by reading the CAN_MSRx register.

The MRTR field in the CAN_MSRx register has no meaning. This field is used only when using Receive and Receive with Overwrite modes.

After a remote frame has been received, the mailbox functions like a transmit mailbox. The message with the highest priority is sent first. The transmitted message may be aborted by setting the MACR bit in the CAN_MCR register. Please refer to the section "Transmission Handling" on page 679.

Figure 33-17. Producer Handling



33.7.3.3.1 Consumer Configuration

A mailbox is in Consumer Mode once the MOT field in the CAN_MMRx register has been configured. Message ID and Message Acceptance masks must be set before Receive Mode is enabled.

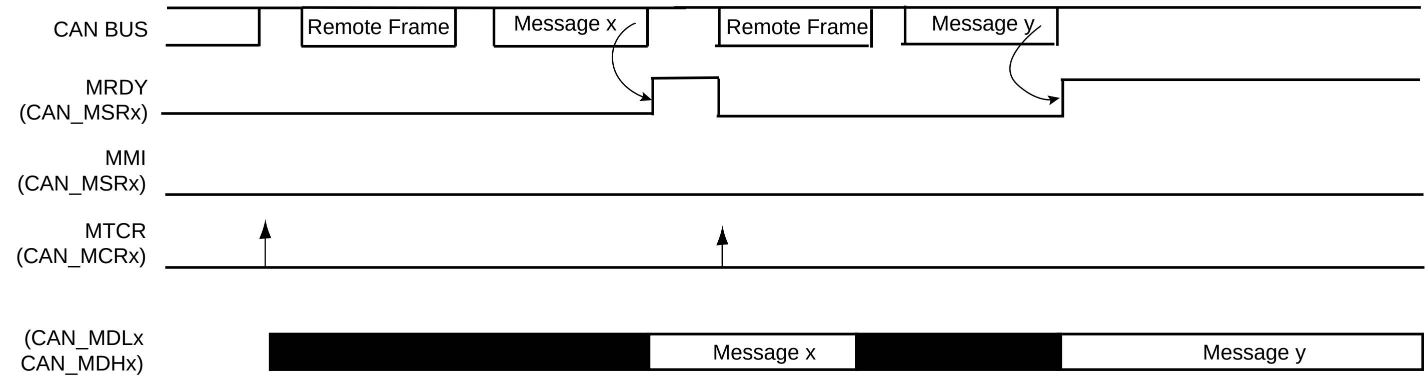
After Consumer Mode is enabled, the MRDY flag in the CAN_MSR register is automatically cleared until the first transfer request command. The software application sends a remote frame by setting the MTCR bit in the CAN_MCRx register or the MBx bit in the global CAN_TCR register. The application is notified of the answer by the MRDY flag set in the CAN_MSRx register. The application can read the data contents in the CAN_MDHx and CAN_MDLx registers. An interrupt is pending for the mailbox while the MRDY flag is set. This interrupt can be masked according to the mailbox flag in the CAN_IMR global register.

The MRTR bit in the CAN_MCRx register has no effect. This field is used only when using Transmit Mode.

After a remote frame has been sent, the consumer mailbox functions as a reception mailbox. The first message received is stored in the mailbox data registers. If other messages intended for this mailbox have been sent while the MRDY flag is set in the CAN_MSRx register, they will be lost. The application is notified by reading the MMI field in the CAN_MSRx register. The read operation automatically clears the MMI flag.

If several messages are answered by the Producer, the CAN controller may have one mailbox in consumer configuration, zero or several mailboxes in Receive Mode and one mailbox in Receive with Overwrite Mode. In this case, the consumer mailbox must have a lower number than the Receive with Overwrite mailbox. The transfer command can be triggered for all mailboxes at the same time by setting several MBx fields in the CAN_TCR register.

Figure 33-18. Consumer Handling



33.7.4 CAN Controller Timing Modes

Using the free running 16-bit internal timer, the CAN controller can be set in one of the two following timing modes:

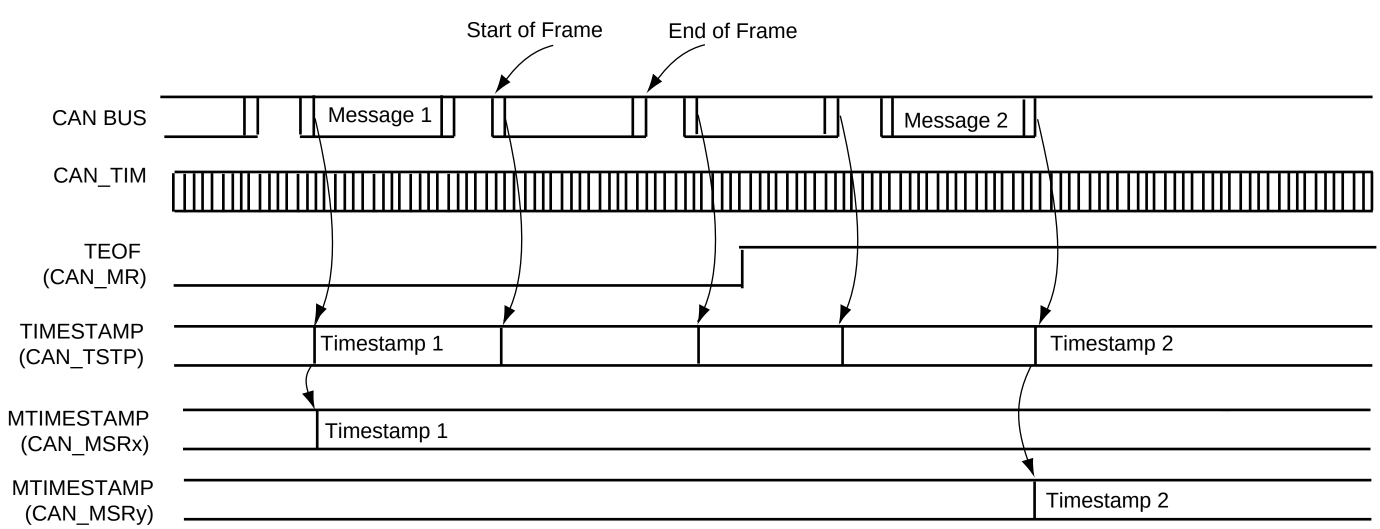
- Timestamping Mode: The value of the internal timer is captured at each Start Of Frame or each End Of Frame.
- Time Triggered Mode: The mailbox transfer operation is triggered when the internal timer reaches the mailbox trigger.

Timestamping Mode is enabled by clearing the TTM bit in the CAN_MR register. Time Triggered Mode is enabled by setting the TTM bit in the CAN_MR register.

33.7.4.1 Timestamping Mode

Each mailbox has its own timestamp value. Each time a message is sent or received by a mailbox, the 16-bit value MTIMESTAMP of the CAN_TIMESTP register is transferred to the LSB bits of the CAN_MSRx register. The value read in the CAN_MSRx register corresponds to the internal timer value at the Start Of Frame or the End Of Frame of the message handled by the mailbox.

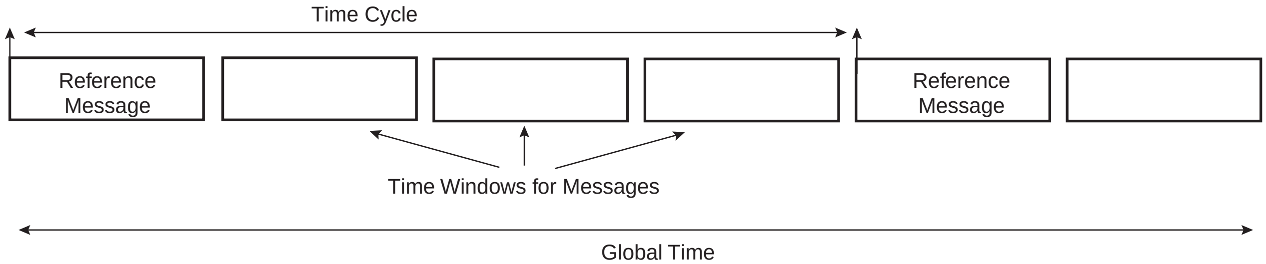
Figure 33-19. Mailbox Timestamp



33.7.4.2 Time Triggered Mode

In Time Triggered Mode, basic cycles can be split into several time windows. A basic cycle starts with a reference message. Each time a window is defined from the reference message, a transmit operation should occur within a pre-defined time window. A mailbox must not win the arbitration in a previous time window, and it must not be retried if the arbitration is lost in the time window.

Figure 33-20. Time Triggered Principle



Time Trigger Mode is enabled by setting the TTM field in the CAN_MR register. In Time Triggered Mode, as in Timestamp Mode, the CAN_TIMESTP field captures the values of the internal counter, but the MTIMESTAMP fields in the CAN_MSRx registers are not active and are read at 0.

33.7.4.2.1 Synchronization by a Reference Message

In Time Triggered Mode, the internal timer counter is automatically reset when a new message is received in the last mailbox. This reset occurs after the reception of the End Of Frame on the rising edge of the MRDY signal in the CAN_MSRx register. This allows synchronization of the internal timer counter with the reception of a reference message and the start a new time window.

33.7.4.2.2 Transmitting within a Time Window

A time mark is defined for each mailbox. It is defined in the 16-bit MTIMEMARK field of the CAN_MMRx register. At each internal timer clock cycle, the value of the CAN_TIM is compared with each mailbox time mark. When the internal timer counter reaches the MTIMEMARK value, an internal timer event for the mailbox is generated for the mailbox.

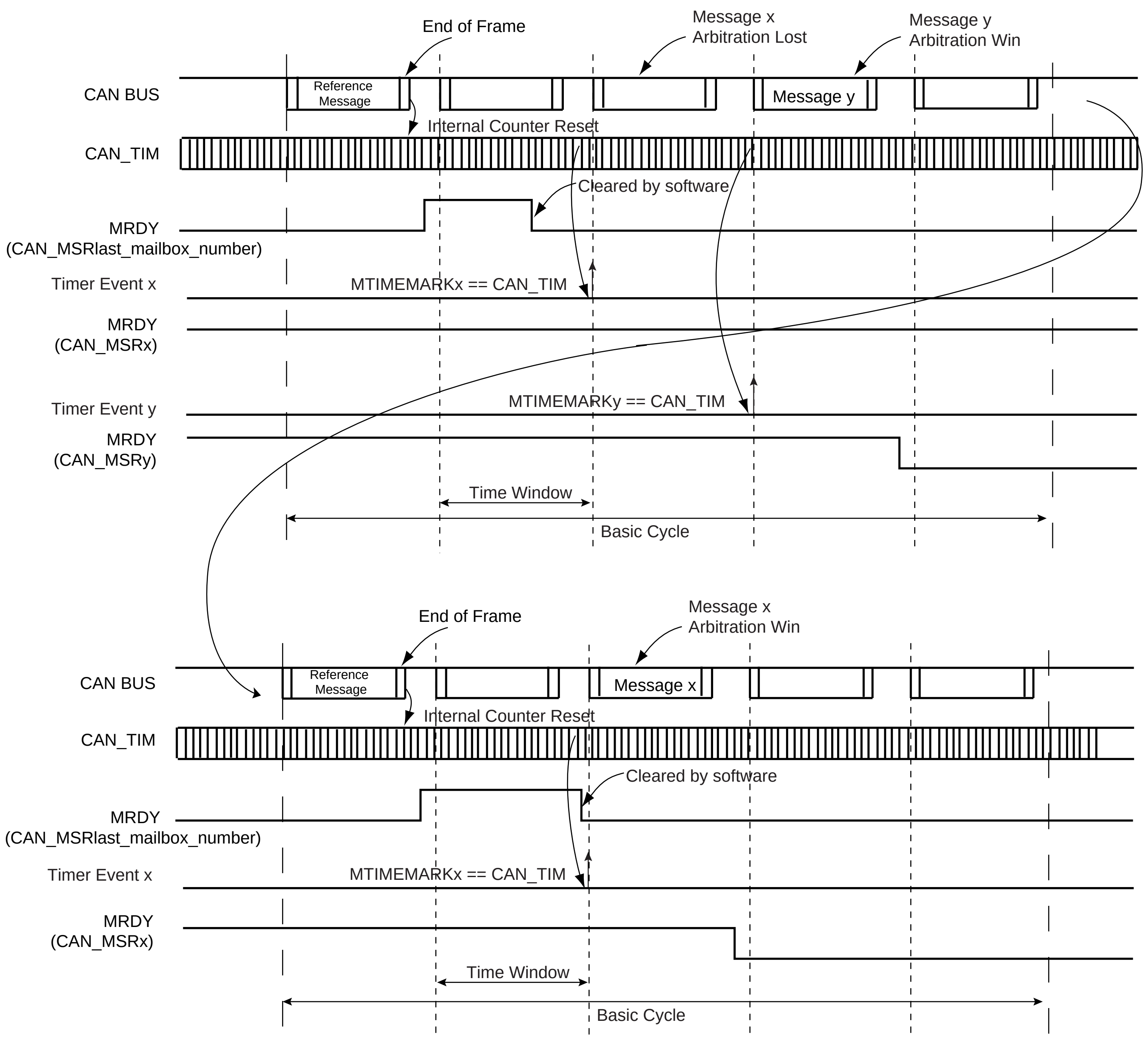
In Time Triggered Mode, transmit operations are delayed until the internal timer event for the mailbox. The application prepares a message to be sent by setting the MTCR in the CAN_MCRx register. The message is not sent until the CAN_TIM value is less than the MTIMEMARK value defined in the CAN_MMRx register.

If the transmit operation is failed, i.e., the message loses the bus arbitration and the next transmit attempt is delayed until the next internal time trigger event. This prevents overlapping the next time window, but the message is still pending and is retried in the next time window when CAN_TIM value equals the MTIMEMARK value. It is also possible to prevent a retry by setting the DRPT field in the CAN_MR register.

33.7.4.2.3 Freezing the Internal Timer Counter

The internal counter can be frozen by setting TIMFRZ in the CAN_MR register. This prevents an unexpected roll-over when the counter reaches FFFFh. When this occurs, it automatically freezes until a new reset is issued, either due to a message received in the last mailbox or any other reset counter operations. The TOVF bit in the CAN_SR register is set when the counter is frozen. The TOVF bit in the CAN_SR register is cleared by reading the CAN_SR register. Depending on the corresponding interrupt mask in the CAN_IMR register, an interrupt is generated when TOVF is set.

Figure 33-21. Time Triggered Operations



33.8 Controller Area Network (CAN) User Interface

Table 33-2. CAN Memory Map

Offset	Register	Name	Access	Reset State
0x0000	Mode Register	CAN_MR	Read-Write	0x0
0x0004	Interrupt Enable Register	CAN_IER	Write-only	-
0x0008	Interrupt Disable Register	CAN_IDR	Write-only	-
0x000C	Interrupt Mask Register	CAN_IMR	Read-only	0x0
0x0010	Status Register	CAN_SR	Read-only	0x0
0x0014	Baudrate Register	CAN_BR	Read/Write	0x0
0x0018	Timer Register	CAN_TIM	Read-only	0x0
0x001C	Timestamp Register	CAN_TIMESTP	Read-only	0x0
0x0020	Error Counter Register	CAN_ECR	Read-only	0x0
0x0024	Transfer Command Register	CAN_TCR	Write-only	-
0x0028	Abort Command Register	CAN_ACR	Write-only	-
0x0100 - 0x01FC	Reserved	–	–	–
0x0200	Mailbox 0 Mode Register	CAN_MMR0	Read/Write	0x0
0x0204	Mailbox 0 Acceptance Mask Register	CAN_MAM0	Read/Write	0x0
0x0208	Mailbox 0 ID Register	CAN_MID0	Read/Write	0x0
0x020C	Mailbox 0 Family ID Register	CAN_MFID0	Read-only	0x0
0x0210	Mailbox 0 Status Register	CAN_MSR0	Read-only	0x0
0x0214	Mailbox 0 Data Low Register	CAN_MDL0	Read/Write	0x0
0x0218	Mailbox 0 Data High Register	CAN_MDH0	Read/Write	0x0
0x021C	Mailbox 0 Control Register	CAN_MCR0	Write-only	-
0x0220	Mailbox 1 Mode Register	CAN_MMR1	Read/Write	0x0
0x0224	Mailbox 1 Acceptance Mask Register	CAN_MAM1	Read/Write	0x0
0x0228	Mailbox 1 ID register	CAN_MID1	Read/Write	0x0
0x022C	Mailbox 1 Family ID Register	CAN_MFID1	Read-only	0x0
0x0230	Mailbox 1 Status Register	CAN_MSR1	Read-only	0x0
0x0234	Mailbox 1 Data Low Register	CAN_MDL1	Read/Write	0x0
0x0238	Mailbox 1 Data High Register	CAN_MDH1	Read/Write	0x0
0x023C	Mailbox 1 Control Register	CAN_MCR1	Write-only	-
...	...	...	...	-

33.8.1 CAN Mode Register

Name: CAN_MR

Access Type: Read/Write

31	30	29	28	27	26	25	24
–	–	–	–	–	RXSYNC		
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	–	–	–	–
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	–	–	–	–	–	–
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
DRPT	TIMFRZ	TTM	TEOF	OVL	ABM	LPM	CANEN

- **CANEN: CAN Controller Enable**

0 = The CAN Controller is disabled.

1 = The CAN Controller is enabled.

- **LPM: Disable/Enable Low Power Mode**

w Power Mode.

1 = Enable Low Power M

CAN controller enters Low Power Mode once all pending messages have been transmitted.

- **ABM: Disable/Enable Autobaud/Listen mode**

0 = Disable Autobaud/listen mode.

1 = Enable Autobaud/listen mode.

- **OVL: Disable/Enable Overload Frame**

0 = No overload frame is generated.

1 = An overload frame is generated after each successful reception for mailboxes configured in Receive with/without overwrite Mode, Producer and Consumer.

- **TEOF: Timestamp messages at each end of Frame**

0 = The value of CAN_TIM is captured in the CAN_TIMESTP register at each Start Of Frame.

1 = The value of CAN_TIM is captured in the CAN_TIMESTP register at each End Of Frame.

- **TTM: Disable/Enable Time Triggered Mode**

0 = Time Triggered Mode is disabled.

1 = Time Triggered Mode is enabled.

- **TIMFRZ: Enable Timer Freeze**

0 = The internal timer continues to be incremented after it reached 0xFFFF.

1 = The internal timer stops incrementing after reaching 0xFFFF. It is restarted after a timer reset. See "Freezing the Internal Timer Counter" on page 684.

- **DRPT: Disable Repeat**

0 = When a transmit mailbox loses the bus arbitration, the transfer request remains pending.

1 = When a transmit mailbox lose the bus arbitration, the transfer request is automatically aborted. It automatically raises the MABT and MRDT flags in the corresponding CAN_MSRx.

- **RXSYNC: Reception Synchronization Stage (not readable)**

This field allows configuration of the reception stage of the macrocell (for debug purposes only)

RXSYNC	Reception Synchronization Stage
0	Rx Signal with Double Synchro Stages (2 Positive Edges)
1	Rx Signal with Double Synchro Stages (One Positive Edge and One Negative Edge)
2	Rx Signal with Single Synchro Stage (Positive Edge)
others	Rx Signal with No Synchro Stage

33.8.2 CAN Interrupt Enable Register

Name: CAN_IER

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR

23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA

15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Enable**

0 = No effect.

1 = Enable Mailbox x interrupt.

- **ERRA: Error Active mode Interrupt Enable**

0 = No effect.

1 = Enable ERRA interrupt.

- **WARN: Warning Limit Interrupt Enable**

0 = No effect.

1 = Enable WARN interrupt.

- **ERRP: Error Passive mode Interrupt Enable**

0 = No effect.

1 = Enable ERRP interrupt.

- **BOFF: Bus-off mode Interrupt Enable**

0 = No effect.

1 = Enable BOFF interrupt.

- **SLEEP: Sleep Interrupt Enable**

0 = No effect.

1 = Enable SLEEP interrupt.

- **WAKEUP: Wakeup Interrupt Enable**

0 = No effect.

1 = Enable SLEEP interrupt.

- **TOVF: Timer Overflow Interrupt Enable**

0 = No effect.

1 = Enable TOVF interrupt.

- **TSTP: TimeStamp Interrupt Enable**

0 = No effect.

1 = Enable TSTP interrupt.

- **CERR: CRC Error Interrupt Enable**

0 = No effect.

1 = Enable CRC Error interrupt.

- **SERR: Stuffing Error Interrupt Enable**

0 = No effect.

1 = Enable Stuffing Error interrupt.

- **AERR: Acknowledgment Error Interrupt Enable**

0 = No effect.

1 = Enable Acknowledgment Error interrupt.

- **FERR: Form Error Interrupt Enable**

0 = No effect.

1 = Enable Form Error interrupt.

- **BERR: Bit Error Interrupt Enable**

0 = No effect.

1 = Enable Bit Error interrupt.

33.8.3 CAN Interrupt Disable Register

Name: CAN_IDR

Access Type: Write-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR

23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA

15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Disable**

0 = No effect.

1 = Disable Mailbox x interrupt.

- **ERRA: Error Active Mode Interrupt Disable**

0 = No effect.

1 = Disable ERRA interrupt.

- **WARN: Warning Limit Interrupt Disable**

0 = No effect.

1 = Disable WARN interrupt.

- **ERRP: Error Passive mode Interrupt Disable**

0 = No effect.

1 = Disable ERRP interrupt.

- **BOFF: Bus-off mode Interrupt Disable**

0 = No effect.

1 = Disable BOFF interrupt.

- **SLEEP: Sleep Interrupt Disable**

0 = No effect.

1 = Disable SLEEP interrupt.

- **WAKEUP: Wakeup Interrupt Disable**

0 = No effect.

1 = Disable WAKEUP interrupt.

- **TOVF: Timer Overflow Interrupt**

0 = No effect.

1 = Disable TOVF interrupt.

- **TSTP: TimeStamp Interrupt Disable**

0 = No effect.

1 = Disable TSTP interrupt.

- **CERR: CRC Error Interrupt Disable**

0 = No effect.

1 = Disable CRC Error interrupt.

- **SERR: Stuffing Error Interrupt Disable**

0 = No effect.

1 = Disable Stuffing Error interrupt.

- **AERR: Acknowledgment Error Interrupt Disable**

0 = No effect.

1 = Disable Acknowledgment Error interrupt.

- **FERR: Form Error Interrupt Disable**

0 = No effect.

1 = Disable Form Error interrupt.

- **BERR: Bit Error Interrupt Disable**

0 = No effect.

1 = Disable Bit Error interrupt.

33.8.4 CAN Interrupt Mask Register

Name: CAN_IMR

Access Type: Read-only

31	30	29	28	27	26	25	24
–	–	–	BERR	FERR	AERR	SERR	CERR

23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA

15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Interrupt Mask**

0 = Mailbox x interrupt is disabled.

1 = Mailbox x interrupt is enabled.

- **ERRA: Error Active mode Interrupt Mask**

0 = ERRA interrupt is disabled.

1 = ERRA interrupt is enabled.

- **WARN: Warning Limit Interrupt Mask**

0 = Warning Limit interrupt is disabled.

1 = Warning Limit interrupt is enabled.

- **ERRP: Error Passive Mode Interrupt Mask**

0 = ERRP interrupt is disabled.

1 = ERRP interrupt is enabled.

- **BOFF: Bus-off Mode Interrupt Mask**

0 = BOFF interrupt is disabled.

1 = BOFF interrupt is enabled.

- **SLEEP: Sleep Interrupt Mask**

0 = SLEEP interrupt is disabled.

1 = SLEEP interrupt is enabled.

- **WAKEUP: Wakeup Interrupt Mask**

0 = WAKEUP interrupt is disabled.

1 = WAKEUP interrupt is enabled.

- **TOVF: Timer Overflow Interrupt Mask**

0 = TOVF interrupt is disabled.

1 = TOVF interrupt is enabled.

- **TSTP: Timestamp Interrupt Mask**

0 = TSTP interrupt is disabled.

1 = TSTP interrupt is enabled.

- **CERR: CRC Error Interrupt Mask**

0 = CRC Error interrupt is disabled.

1 = CRC Error interrupt is enabled.

- **SERR: Stuffing Error Interrupt Mask**

0 = Bit Stuffing Error interrupt is disabled.

1 = Bit Stuffing Error interrupt is enabled.

- **AERR: Acknowledgment Error Interrupt Mask**

0 = Acknowledgment Error interrupt is disabled.

1 = Acknowledgment Error interrupt is enabled.

- **FERR: Form Error Interrupt Mask**

0 = Form Error interrupt is disabled.

1 = Form Error interrupt is enabled.

- **BERR: Bit Error Interrupt Mask**

0 = Bit Error interrupt is disabled.

1 = Bit Error interrupt is enabled.

33.8.5 CAN Status Register

Name: CAN_SR

Access Type: Read-only

31	30	29	28	27	26	25	24
OVLSY	TBSY	RBSY	BERR	FERR	AERR	SERR	CERR

23	22	21	20	19	18	17	16
TSTP	TOVF	WAKEUP	SLEEP	BOFF	ERRP	WARN	ERRA

15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

- **MBx: Mailbox x Event**

0 = No event occurred on Mailbox x.

1 = An event occurred on Mailbox x.

An event corresponds to MRDY, MABT fields in the CAN_MSRx register.

- **ERRA: Error Active mode**

0 = CAN controller is not in error active mode

1 = CAN controller is in error active mode

This flag is set depending on TEC and REC counter values. It is set when node is neither in error passive mode nor in bus off mode.

This flag is automatically reset when above condition is not satisfied.

- **WARN: Warning Limit**

0 = CAN controller Warning Limit is not reached.

1 = CAN controller Warning Limit is reached.

This flag is set depending on TEC and REC counters values. It is set when at least one of the counters values exceeds 96.

This flag is automatically reset when above condition is not satisfied.

- **ERRP: Error Passive mode**

0 = CAN controller is not in error passive mode

1 = CAN controller is in error passive mode

This flag is set depending on TEC and REC counters values.

A node is error passive when TEC counter is greater or equal to 128 (decimal) or when the REC counter is greater or equal to 128 (decimal) and less than 256.

This flag is automatically reset when above condition is not satisfied.

- **BOFF: Bus Off mode**

0 = CAN controller is not in bus-off mode

1 = CAN controller is in bus-off mode

This flag is set depending on TEC counter value. A node is bus off when TEC counter is greater or equal to 256 (decimal).

This flag is automatically reset when above condition is not satisfied.

- **SLEEP: CAN controller in Low power Mode**

0 = CAN controller is not in low power mode.

1 = CAN controller is in low power mode.

This flag is automatically reset when Low power mode is disabled

- **WAKEUP: CAN controller is not in Low power Mode**

0 = CAN controller is in low power mode.

1 = CAN controller is not in low power mode.

When a WAKEUP event occurs, the CAN controller is synchronized with the bus activity. Messages can be transmitted or received. The CAN controller clock must be available when a WAKEUP event occurs. This flag is automatically reset when the CAN Controller enters Low Power mode.

- **TOVF: Timer Overflow**

0 = The timer has not rolled-over FFFFh to 0000h.

1 = The timer rolls-over FFFFh to 0000h.

This flag is automatically cleared by reading CAN_SR register.

- **TSTP Timestamp**

0 = No bus activity has been detected.

1 = A start of frame or an end of frame has been detected (according to the TEOF field in the CAN_MR register).

This flag is automatically cleared by reading the CAN_SR register.

- **CERR: Mailbox CRC Error**

0 = No CRC error occurred during a previous transfer.

1 = A CRC error occurred during a previous transfer.

A CRC error has been detected during last reception.

This flag is automatically cleared by reading CAN_SR register.

- **SERR: Mailbox Stuffing Error**

0 = No stuffing error occurred during a previous transfer.

1 = A stuffing error occurred during a previous transfer.

A form error results from the detection of more than five consecutive bit with the same polarity.

This flag is automatically cleared by reading CAN_SR register.

- **AERR: Acknowledgment Error**

0 = No acknowledgment error occurred during a previous transfer.

1 = An acknowledgment error occurred during a previous transfer.

An acknowledgment error is detected when no detection of the dominant bit in the acknowledge slot occurs.

This flag is automatically cleared by reading CAN_SR register.

- **FERR: Form Error**

0 = No form error occurred during a previous transfer

1 = A form error occurred during a previous transfer

A form error results from violations on one or more of the fixed form of the following bit fields:

- – CRC delimiter
- – ACK delimiter
- – End of frame
- – Error delimiter
- – Overload delimiter

This flag is automatically cleared by reading CAN_SR register.

- **BERR: Bit Error**

0 = No bit error occurred during a previous transfer.

1 = A bit error occurred during a previous transfer.

A bit error is set when the bit value monitored on the line is different from the bit value sent.

This flag is automatically cleared by reading CAN_SR register.

- **RBSY: Receiver busy**

0 = CAN receiver is not receiving a frame.

1 = CAN receiver is receiving a frame.

Receiver busy. This status bit is set by hardware while CAN receiver is acquiring or monitoring a frame (remote, data, overload or error frame). It is automatically reset when CAN is not receiving.

- **TBSY: Transmitter busy**

0 = CAN transmitter is not transmitting a frame.

1 = CAN transmitter is transmitting a frame.

Transmitter busy. This status bit is set by hardware while CAN transmitter is generating a frame (remote, data, overload or error frame). It is automatically reset when CAN is not transmitting.

- **OVLSY: Overload busy**

0 = CAN transmitter is not transmitting an overload frame.

1 = CAN transmitter is transmitting a overload frame.

It is automatically reset when the bus is not transmitting an overload frame.

33.8.6 CAN Baudrate Register

Name: CAN_BR

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
–	BRP						

15	14	13	12	11	10	9	8
–	–	SJW		–	PROPAG		

7	6	5	4	3	2	1	0
–	PHASE1			–	PHASE2		

Any modification on one of the fields of the CANBR register must be done while CAN module is disabled.

To compute the different Bit Timings, please refer to the Section 33.6.4.1 "CAN Bit Timing Configuration" on page 666.

- **PHASE2: Phase 2 segment**

This phase is used to compensate the edge phase error.

$$t_{PHS2} = t_{CSC} \times (PHASE2 + 1)$$

Warning: PHASE2 value must be different from 0.

- **PHASE1: Phase 1 segment**

This phase is used to compensate for edge phase error.

$$t_{PHS1} = t_{CSC} \times (PHASE1 + 1)$$

- **PROPAG: Programming time segment**

This part of the bit time is used to compensate for the physical delay times within the network.

$$t_{PRS} = t_{CSC} \times (PROPAG + 1)$$

- **SJW: Re-synchronization jump width**

To compensate for phase shifts between clock oscillators of different controllers on bus. The controller must re-synchronize on any relevant signal edge of the current transmission. The synchronization jump width defines the maximum of clock cycles a bit period may be shortened or lengthened by re-synchronization.

$$t_{SJW} = t_{CSC} \times (SJW + 1)$$

- **BRP: Baudrate Prescaler.**

This field allows user to program the period of the CAN system clock to determine the individual bit timing.

$$t_{CSC} = (BRP + 1)/MCK$$

The BRP field must be within the range [1, 0x7F], i.e., BRP = 0 is not authorized.

- **SMP: Sampling Mode**

0 = The incoming bit stream is sampled once at sample point.

1 = The incoming bit stream is sampled three times with a period of a MCK clock period, centered on sample point.

SMP Sampling Mode is automatically disabled if BRP = 0.

33.8.7 CAN Timer Register

Name: CAN_TIM

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
TIMER15	TIMER14	TIMER13	TIMER12	TIMER11	TIMER10	TIMER9	TIMER8

7	6	5	4	3	2	1	0
TIMER7	TIMER6	TIMER5	TIMER4	TIMER3	TIMER2	TIMER1	TIMER0

- **TIMERx: Timer**

This field represents the internal CAN controller 16-bit timer value.

33.8.8 CAN Timestamp Register

Name: CAN_TIMESTP

Access Type: Read-only

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

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

15	14	13	12	11	10	9	8
MTIMESTAMP 15	MTIMESTAMP 14	MTIMESTAMP 13	MTIMESTAMP 12	MTIMESTAMP 11	MTIMESTAMP 10	MTIMESTAMP 9	MTIMESTAMP 8

7	6	5	4	3	2	1	0
MTIMESTAMP 7	MTIMESTAMP 6	MTIMESTAMP 5	MTIMESTAMP 4	MTIMESTAMP 3	MTIMESTAMP 2	MTIMESTAMP 1	MTIMESTAMP 0

- **MTIMESTAMPx: Timestamp**

This field represents the internal CAN controller 16-bit timer value.

If the TEOF bit is cleared in the CAN_MR register, the internal Timer Counter value is captured in the MTIMESTAMP field at each start of frame. Else the value is captured at each end of frame. When the value is captured, the TSTP flag is set in the CAN_SR register. If the TSTP mask in the CAN_IMR register is set, an interrupt is generated while TSTP flag is set in the CAN_SR register. This flag is cleared by reading the CAN_SR register.

Note: The CAN_TIMESTP register is reset when the CAN is disabled then enabled thanks to the CANEN bit in the CAN_MR.

33.8.9 CAN Error Counter Register

Name: CAN_ECR

Access Type: Read-only

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

23	22	21	20	19	18	17	16
TEC							

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
REC							

- **REC: Receive Error Counter**

When a receiver detects an error, REC will be increased by one, except when the detected error is a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG.

When a receiver detects a dominant bit as the first bit after sending an ERROR FLAG, REC is increased by 8.

When a receiver detects a BIT ERROR while sending an ACTIVE ERROR FLAG, REC is increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive dominant bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive dominant bits, each receiver increases its REC by 8.

After successful reception of a message, REC is decreased by 1 if it was between 1 and 127. If REC was 0, it stays 0, and if it was greater than 127, then it is set to a value between 119 and 127.

- **TEC: Transmit Error Counter**

When a transmitter sends an ERROR FLAG, TEC is increased by 8 except when

  – the transmitter is error passive and detects an ACKNOWLEDGMENT ERROR because of not detecting a dominant ACK and does not detect a dominant bit while sending its PASSIVE ERROR FLAG.

  – the transmitter sends an ERROR FLAG because a STUFF ERROR occurred during arbitration and should have been recessive and has been sent as recessive but monitored as dominant.

When a transmitter detects a BIT ERROR while sending an ACTIVE ERROR FLAG or an OVERLOAD FLAG, the TEC will be increased by 8.

Any node tolerates up to 7 consecutive dominant bits after sending an ACTIVE ERROR FLAG, PASSIVE ERROR FLAG or OVERLOAD FLAG. After detecting the 14th consecutive dominant bit (in case of an ACTIVE ERROR FLAG or an OVERLOAD FLAG) or after detecting the 8th consecutive dominant bit following a PASSIVE ERROR FLAG, and after each sequence of additional eight consecutive dominant bits every transmitter increases its TEC by 8.

After a successful transmission the TEC is decreased by 1 unless it was already 0.

33.8.10 CAN Transfer Command Register

Name: CAN_TCR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several transfer requests at the same time.

- **MBx: Transfer Request for Mailbox x**

Mailbox Object Type	Description
Receive	It receives the next message.
Receive with overwrite	This triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a consumer.

This flag clears the MRDY and MABT flags in the corresponding CAN_MSRx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn, starting with the mailbox with the highest priority. If several mailboxes have the same priority, then the mailbox with the lowest number is sent first (i.e., MB0 will be transferred before MB1).

- **TIMRST: Timer Reset**

Resets the internal timer counter. If the internal timer counter is frozen, this command automatically re-enables it. This command is useful in Time Triggered mode.

33.8.11 CAN Abort Command Register

Name: CAN_ACR

Access Type: Write-only

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

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

15	14	13	12	11	10	9	8
MB15	MB14	MB13	MB12	MB11	MB10	MB9	MB8

7	6	5	4	3	2	1	0
MB7	MB6	MB5	MB4	MB3	MB2	MB1	MB0

This register initializes several abort requests at the same time.

- **MBx: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame is not serviced.

It is possible to set MACR field (in the CAN_MCRx register) for each mailbox.

33.8.12 CAN Message Mode Register

Name: CAN_MMRx

Access Type: Read/Write

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

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

15	14	13	12	11	10	9	8
MTIMEMARK15	MTIMEMARK14	MTIMEMARK13	MTIMEMARK12	MTIMEMARK11	MTIMEMARK10	MTIMEMARK9	MTIMEMARK8

7	6	5	4	3	2	1	0
MTIMEMARK7	MTIMEMARK6	MTIMEMARK5	MTIMEMARK4	MTIMEMARK3	MTIMEMARK2	MTIMEMARK1	MTIMEMARK0

- **MTIMEMARK: Mailbox Timemark**

This field is active in Time Triggered Mode. Transmit operations are allowed when the internal timer counter reaches the Mailbox Timemark. See "Transmitting within a Time Window" on page 684.

In Timestamp Mode, MTIMEMARK is set to 0.

- **PRIOR: Mailbox Priority**

This field has no effect in receive and receive with overwrite modes. In these modes, the mailbox with the lowest number is serviced first.

When several mailboxes try to transmit a message at the same time, the mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 is serviced before MBx 15 if they have the same priority).

- **MOT: Mailbox Object Type**

This field allows the user to define the type of the mailbox. All mailboxes are independently configurable. Five different types are possible for each mailbox:

MOT			Mailbox Object Type
0	0	0	Mailbox is disabled. This prevents receiving or transmitting any messages with this mailbox.
0	0	1	Reception Mailbox. Mailbox is configured for reception. If a message is received while the mailbox data register is full, it is discarded.
0	1	0	Reception mailbox with overwrite. Mailbox is configured for reception. If a message is received while the mailbox is full, it overwrites the previous message.
0	1	1	Transmit mailbox. Mailbox is configured for transmission.
1	0	0	Consumer Mailbox. Mailbox is configured in reception but behaves as a Transmit Mailbox, i.e., it sends a remote frame and waits for an answer.
1	0	1	Producer Mailbox. Mailbox is configured in transmission but also behaves like a reception mailbox, i.e., it waits to receive a Remote Frame before sending its contents.
1	1	X	Reserved

33.8.13 CAN Message Acceptance Mask Register

Name: CAN_MAMx

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
MIDvA						MIDvB	

15	14	13	12	11	10	9	8
MIDvB							

7	6	5	4	3	2	1	0
MIDvB							

To prevent concurrent access with the internal CAN core, the application must disable the mailbox before writing to CAN_MAMx registers.

- **MIDvB: Complementary bits for identifier in extended frame mode**

Acceptance mask for corresponding field of the message IDvB register of the mailbox.

- **MIDvA: Identifier for standard frame mode**

Acceptance mask for corresponding field of the message IDvA register of the mailbox.

- **MIDE: Identifier Version**

0= Compares IDvA from the received frame with the CAN_MIDx register masked with CAN_MAMx register.

1= Compares IDvA and IDvB from the received frame with the CAN_MIDx register masked with CAN_MAMx register.

33.8.14 CAN Message ID Register

Name: CAN_MIDx

Access Type: Read/Write

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

23	22	21	20	19	18	17	16
MIDvA						MIDvB	

15	14	13	12	11	10	9	8
MIDvB							

7	6	5	4	3	2	1	0
MIDvB							

To prevent concurrent access with the internal CAN core, the application must disable the mailbox before writing to CAN_MIDx registers.

- **MIDvB: Complementary bits for identifier in extended frame mode**

If MIDE is cleared, MIDvB value is 0.

- **MIDE: Identifier Version**

This bit allows the user to define the version of messages processed by the mailbox. If set, mailbox is dealing with version 2.0 Part B messages; otherwise, mailbox is dealing with version 2.0 Part A messages.

- **MIDvA: Identifier for standard frame mode**

33.8.15 CAN Message Family ID Register

Name: CAN_MFIDx

Access Type: Read-only

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

23	22	21	20	19	18	17	16
MFID							

15	14	13	12	11	10	9	8
MFID							

7	6	5	4	3	2	1	0
MFID							

- **MFID: Family ID**

This field contains the concatenation of CAN_MIDx register bits masked by the CAN_MAMx register. This field is useful to speed up message ID decoding. The message acceptance procedure is described below.

As an example:

```
CAN_MIDx = 0x305A4321
CAN_MAMx = 0x3FF0F0FF
CAN_MFIDx = 0x000000A3
```

33.8.16 CAN Message Status Register

Name: CAN_MSRx

Access Type: Read only

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

23	22	21	20	19	18	17	16
MRDY	MABT	–	MRTR	MDLC			

15	14	13	12	11	10	9	8
MTIMESTAMP 15	MTIMESTAMP 14	MTIMESTAMP 13	MTIMESTAMP 12	MTIMESTAMP 11	MTIMESTAMP 10	MTIMESTAMP 9	MTIMESTAMP 8

7	6	5	4	3	2	1	0
MTIMESTAMP 7	MTIMESTAMP 6	MTIMESTAMP 5	MTIMESTAMP 4	MTIMESTAMP 3	MTIMESTAMP 2	MTIMESTAMP 1	MTIMESTAMP 0

These register fields are updated each time a message transfer is received or aborted.

MMI is cleared by reading the CAN_MSRx register.

MRDY, MABT are cleared by writing MTCR or MACR in the CAN_MCRx register.

Warning: MRTR and MDLC state depends partly on the mailbox object type.

- **MTIMESTAMP: Timer value**

This field is updated only when time-triggered operations are disabled (TTM cleared in CAN_MR register). If the TEOF field in the CAN_MR register is cleared, TIMESTAMP is the internal timer value at the start of frame of the last message received or sent by the mailbox. If the TEOF field in the CAN_MR register is set, TIMESTAMP is the internal timer value at the end of frame of the last message received or sent by the mailbox.

In Time Triggered Mode, MTIMESTAMP is set to 0.

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	Length of the first mailbox message received
Receive with overwrite	Length of the last mailbox message received
Transmit	No action
Consumer	Length of the mailbox message received
Producer	Length of the mailbox message to be sent after the remote frame reception

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	The first frame received has the RTR bit set.
Receive with overwrite	The last frame received has the RTR bit set.
Transmit	Reserved
Consumer	Reserved. After setting the MOT field in the CAN_MMR, MRTR is reset to 1.
Producer	Reserved. After setting the MOT field in the CAN_MMR, MRTR is reset to 0.

- **MABT: Mailbox Message Abort**

An interrupt is triggered when MABT is set.

0 = Previous transfer is not aborted.

1 = Previous transfer has been aborted.

This flag is cleared by writing to CAN_MCRx register

Mailbox Object Type	Description
Receive	Reserved
Receive with overwrite	Reserved
Transmit	Previous transfer has been aborted
Consumer	The remote frame transfer request has been aborted.
Producer	The response to the remote frame transfer has been aborted.

- **MRDY: Mailbox Ready**

An interrupt is triggered when MRDY is set.

0 = Mailbox data registers can not be read/written by the software application. CAN_MDx are locked by the CAN_MDx.

1 = Mailbox data registers can be read/written by the software application.

This flag is cleared by writing to CAN_MCRx register.

Mailbox Object Type	Description
Receive	At least one message has been received since the last mailbox transfer order. Data from the first frame received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Receive with overwrite	At least one frame has been received since the last mailbox transfer order. Data from the last frame received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Transmit	Mailbox data have been transmitted. After setting the MOT field in the CAN_MMR, MRDY is reset to 1.
Consumer	At least one message has been received since the last mailbox transfer order. Data from the first message received can be read in the CAN_MDxx registers. After setting the MOT field in the CAN_MMR, MRDY is reset to 0.
Producer	A remote frame has been received, mailbox data have been transmitted. After setting the MOT field in the CAN_MMR, MRDY is reset to 1.

- **MMI: Mailbox Message Ignored**

0 = No message has been ignored during the previous transfer

1 = At least one message has been ignored during the previous transfer

Cleared by reading the CAN_MSRx register.

Mailbox Object Type	Description
Receive	Set when at least two messages intended for the mailbox have been sent. The first one is available in the mailbox data register. Others have been ignored. A mailbox with a lower priority may have accepted the message.
Receive with overwrite	Set when at least two messages intended for the mailbox have been sent. The last one is available in the mailbox data register. Previous ones have been lost.
Transmit	Reserved
Consumer	A remote frame has been sent by the mailbox but several messages have been received. The first one is available in the mailbox data register. Others have been ignored. Another mailbox with a lower priority may have accepted the message.
Producer	A remote frame has been received, but no data are available to be sent.

33.8.17 CAN Message Data Low Register

Name: CAN_MDLx

Access Type: Read/Write

31	30	29	28	27	26	25	24
MDL							

23	22	21	20	19	18	17	16
MDL							

15	14	13	12	11	10	9	8
MDL							

7	6	5	4	3	2	1	0
MDL							

- **MDL: Message Data Low Value**

When MRDY field is set in the CAN_MSRx register, the lower 32 bits of a received message can be read or written by the software application. Otherwise, the MDL value is locked by the CAN controller to send/receive a new message.

In Receive with overwrite, the CAN controller may modify MDL value while the software application reads MDH and MDL registers. To check that MDH and MDL do not belong to different messages, the application has to check the MMI field in the CAN_MSRx register. In this mode, the software application must re-read CAN_MDH and CAN_MDL, while the MMI bit in the CAN_MSRx register is set.

Bytes are received/sent on the bus in the following order:

1. CAN_MDL[7:0]
2. CAN_MDL[15:8]
3. CAN_MDL[23:16]
4. CAN_MDL[31:24]
5. CAN_MDH[7:0]
6. CAN_MDH[15:8]
7. CAN_MDH[23:16]
8. CAN_MDH[31:24]

33.8.18 CAN Message Data High Register

Name: CAN_MDHx

Access Type: Read/Write

31	30	29	28	27	26	25	24
MDH							

23	22	21	20	19	18	17	16
MDH							

15	14	13	12	11	10	9	8
MDH							

7	6	5	4	3	2	1	0
MDH							

- **MDH: Message Data High Value**

When MRDY field is set in the CAN_MSRx register, the upper 32 bits of a received message are read or written by the software application. Otherwise, the MDH value is locked by the CAN controller to send/receive a new message.

In Receive with overwrite, the CAN controller may modify MDH value while the software application reads MDH and MDL registers. To check that MDH and MDL do not belong to different messages, the application has to check the MMI field in the CAN_MSRx register. In this mode, the software application must re-read CAN_MDH and CAN_MDL, while the MMI bit in the CAN_MSRx register is set.

Bytes are received/sent on the bus in the following order:

1. CAN_MDL[7:0]
2. CAN_MDL[15:8]
3. CAN_MDL[23:16]
4. CAN_MDL[31:24]
5. CAN_MDH[7:0]
6. CAN_MDH[15:8]
7. CAN_MDH[23:16]
8. CAN_MDH[31:24]

33.8.19 CAN Message Control Register

Name: CAN_MCRx

Access Type: Write-only

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

23	22	21	20	19	18	17	16
MTCR	MACR	–	MRTR	MDLC			

15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–

7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

- **MDLC: Mailbox Data Length Code**

Mailbox Object Type	Description
Receive	No action.
Receive with overwrite	No action.
Transmit	Length of the mailbox message.
Consumer	No action.
Producer	Length of the mailbox message to be sent after the remote frame reception.

- **MRTR: Mailbox Remote Transmission Request**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Set the RTR bit in the sent frame
Consumer	No action, the RTR bit in the sent frame is set automatically
Producer	No action

Consumer situations can be handled automatically by setting the mailbox object type in Consumer. This requires only one mailbox.

It can also be handled using two mailboxes, one in reception, the other in transmission. The MRTR and the MTCR bits must be set in the same time.

- **MACR: Abort Request for Mailbox x**

Mailbox Object Type	Description
Receive	No action
Receive with overwrite	No action
Transmit	Cancels transfer request if the message has not been transmitted to the CAN transceiver.
Consumer	Cancels the current transfer before the remote frame has been sent.
Producer	Cancels the current transfer. The next remote frame will not be serviced.

It is possible to set MACR field for several mailboxes in the same time, setting several bits to the CAN_ACR register.

- **MTCR: Mailbox Transfer Command**

Mailbox Object Type	Description
Receive	Allows the reception of the next message.
Receive with overwrite	Triggers a new reception.
Transmit	Sends data prepared in the mailbox as soon as possible.
Consumer	Sends a remote transmission frame.
Producer	Sends data prepared in the mailbox after receiving a remote frame from a Consumer.

This flag clears the MRDY and MABT flags in the CAN_MSRx register.

When several mailboxes are requested to be transmitted simultaneously, they are transmitted in turn. The mailbox with the highest priority is serviced first. If several mailboxes have the same priority, the mailbox with the lowest number is serviced first (i.e., MBx0 will be serviced before MBx 15 if they have the same priority).

It is possible to set MTCR for several mailboxes at the same time by writing to the CAN_TCR register.

34. Electrical Characteristics

34.1 Absolute Maximum Ratings

Table 34-1. Absolute Maximum Ratings*

Operating Temperature (Industrial)	-40°C to +125°C
Storage Temperature	-60°C to +150°C
Voltage on Input Pins with Respect to Ground	-0.3V to +4.0V
Maximum Operating Voltage (VDDCORE, VDDOSCS, VDDOSCM)	1.5V
Maximum Operating Voltage (VDDPLLA, VDDIOMP, VDDIOM and VDDIOP)	4.0V
Total DC Output Current on all I/O lines	350mA

*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.

34.2 DC Characteristics

The following characteristics are applicable to the operating temperature range: T_A = -40°C to 85°C, unless otherwise specified.

Table 34-2. DC Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Units
$V_{VDDCORE}$	DC Supply Core @ 1.1V		1.05	1.1	1.15	V
	DC Supply Core @1.2V		1.14	1.2	1.26	V
$V_{VDDOSCS}$	DC Supply 32K Oscillator @ 1.1V		1.05	1.1	1.15	V
	DC Supply 32K Oscillator @ 1.2V		1.14	1.2	1.26	V
$V_{VDDOSCM}$	DC Supply Main Oscillator and PLLB @ 1.1V		1.05	1.1	1.15	
	DC Supply Main Oscillator and PLLB @ 1.2V		1.14	1.2	1.26	V
$V_{VDDPLLA}$	DC Supply PLLA		3.0	3.3	3.6	V
V_{VDDIOM}	DC Supply Memory I/Os		1.65	1.8	1.95	V
			3.0	3.3	3.6	V
$V_{VDDIOMP}$	DC Supply Memory/Peripheral I/Os		1.65	1.8	1.95	V
			3.0	3.3	3.6	V
V_{VDDIOP}	DC Supply Peripheral I/Os		3.0		3.6	V
V_{IL}	Input Low-level Voltage		-0.3		0.8	V
V_{IH}	Input High-level Voltage	V_{VDDIO}= V_{VDDIOM} or V_{VDDIOP}	2		V_{VDDIO}+0.3	V
V_{OL}	Output Low-level Voltage				0.4	V

Table 34-2. DC Characteristics (Continued)

V_{OH}	Output High-level Voltage	V_{VDDIO}= V_{VDDIOM} or V_{VDDIOP}		V_{VDDIO}-0.4			V
I_{LEAK}	Input Leakage Current	Pullup resistors disabled				±1	µA
C_{IN}	Input Capacitance	324-ball CABGA Package				5.0 TBC	pF
R_{PULLUP}	Pull-up Resistance	PIOA0-PIOA31, PIOB0-PIOB31, PIOC0-PIOC31		70	100	175	kOhm
I_O	Output Current	PIOA0-PIOA31, PIOB0-PIOB31, PIOC0-PC31				8	mA
I_{SC}	Static Current	MCK = 0 Hz, excluding POR	T_A =25°C		TBD		µA
		All inputs driven A_TMS, A_TDI, A_TCK, A_NRST = 1	T_A =85°C			TBD	

34.3 Power Consumption

- Power consumption of power supply in three different modes: Full Speed, Idle Mode and Quasi Static.
- Power consumption by peripheral: calculated as the difference in current measurement after having first enabled and then disabled the corresponding clock.

34.3.1 Power Consumption versus Modes

The values in Table 34-3 and Table 34-4 on page 717 are measured values of the power consumption with the following operating conditions:

- V_{DDIOM} = V_{DDIOP} = V_{DDIOMP} =V_{DDPLLA} = 3.3V
- V_{DDOSCM} = $V_{DDOSC32}$ =1.2V
- There is no consumption on the I/Os of the device.

All measurement refer to VDDCore supply.

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

Table 34-3. Power Consumption for Different Modes

Mode	Conditions	Consumption	Unit
Full speed	ARM Core clock is 200 MHz. MCK is 100 MHz. Dhrystone running in ARM Icache. FFT running i MagicV PM $V_{DDCORE} = 1.2V$ $T_A = 25°C$	TBD	mA
Idle[(1)]	MCK is 96 MHz. ARM core in idle state, waiting for an interrupt. Processor, MagicV and peripherals clock disabled $V_{DDCORE} = 1.2V$ $T_A = 25°C$	TBD	mA
Quasi Static	ARM Core clock is 500 Hz. MCK is 500 Hz Processor, MagicV and peripherals clock disabled $V_{DDCORE} = 1.2V$ $T_A = 25°C$	TBD	µA

Note: 1. No SRAM access in Idle Mode.

Table 34-4. Power Consumption by Peripheral (TA = 25°C, VDDCORE = 1.2V)

Peripheral	Consumption	Unit
PIO Controller	4.5 (TBC)	µA/MHz
USART	1.7 (TBC)	
UHP	12.1 (TBC)	
UDP	8.9 (TBC)	
CAN	TBD	
TWI	2.1 (TBC)	
SPI	9.5 (TBC)	
MCI	12.9 (TBC)	
SSC	15.3 (TBC)	
ETH MAC	TBD	
Timer Counter Channels	3.0 (TBC)	

34.4 Clock Characteristics

34.4.1 Processor Clock Characteristics

Table 34-5. ARM Clock Waveform Parameters

Symbol	Parameter	Conditions	Min	Max	Units
$1/(t_{CPPCK})$	ARM Clock Frequency @1.1V	VDDCORE = 1.05V T = 70°C		160	MHz
$1/(t_{CPPCK})$	ARM Clock Frequency @1.2V	VDDCORE = 1.08V T = 85°C TCM access enabled		180 (TBC)	MHz
$1/(t_{CPPCK})$	ARM Clock Frequency @1.2V	VDDCORE = 1.08V T = 85°C TCM access disabled		200 (TBC)	MHz

34.4.2 XIN Clock Characteristics

Table 34-6. 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}	

Note: 1. These characteristics apply only when the Main Oscillator is in bypass mode (i.e., when MOSCEN = 0 and OSCBYPASS = 1 in the CKGR_MOR register.)

34.5 Crystal Oscillator Characteristics

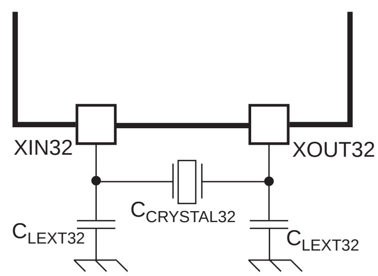
The following characteristics are applicable to the operating temperature range: T_A = -40°C to 85°C and to power supply worst case, unless otherwise specified.

34.5.1 32 kHz Oscillator Characteristics

Table 34-7. 32 kHz Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CP32KHz})$	Crystal Oscillator Frequency			32.768		kHz
$C_{CRYSTAL32}$	Crystal Load Capacitance	Crystal @ 32.768 kHz	6		12.5	pF
C_{LEXT32} (2)	External Load Capacitance	$C_{CRYSTAL32}$ = 6pF(3)		4		pF
		$C_{CRYSTAL32}$ = 12.5pF(3)		17		pF
	Duty Cycle		40		60	%
t_{ST}	Startup Time	$V_{DDOSC32}$ = 1.2V R_S = 50 kΩ, C_L = 6pF(1)			300	ms
		$V_{DDOSC32}$ = 1.2V R_S = 50 kΩ, C_L = 12.5 pF(1)			900	ms
		$V_{DDOSC32}$ = 1.2V R_S = 100 kΩ, C_L = 6pF(1)			600	ms
		$V_{DDOSC32}$ = 1.2V R_S = 100 kΩ, C_L = 12.5 pF(1)			1200	ms

Notes:
1. R_S is the equivalent series resistance, C_L is the equivalent load capacitance.
2. C_{LEXT32} is determined by taking into account internal parasitic and package load capacitance.
3. Additional user load capacitance should be subtracted from C_{LEXT32}.

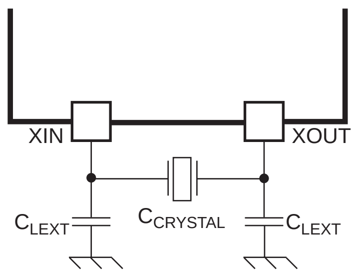


34.5.2 Main Oscillator Characteristics

Table 34-8. Main Oscillator Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$1/(t_{CPMAIN})$	Crystal Oscillator Frequency		8		16	MHz
$C_{CRYSTAL}$	Crystal Load Capacitance		15		20	pF
C_{LEXT} [4]	External Load Capacitance	$C_{CRYSTAL}$ = 15 pF[3]		19		pF
		$C_{CRYSTAL}$ = 20 pF		29		pF
	Duty Cycle		40	50	60	%
t_{ST}	Startup Time	V_{DDOSCM} = 1.08 to 1.32V C_S = 7 pF[1] $1/(t_{CPMAIN})$ = 16 MHz			2	ms
I_{DDST}	Standby Current Consumption	Standby mode			1	µA
P_{ON}	Drive Level	@ 16 MHz			150	µW
$I_{DD\ ON}$	Current Dissipation	@ 16 MHz[2]		300	530	µA

Notes: 1. C_S is the shunt capacitance.
2. R_S = 25 to 50 Ω ; C_S = 2.5 to 3.0 pF; C_M = 7 to 5 fF (typ, worst case).
3. Additional user load capacitance should be subtracted from C_{LEXT}.
4. C_{LEXT} is determined by taking into account internal parasitic and package load capacitance.



34.5.3 Crystal Characteristics

Table 34-9. Crystal Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
ESR	Equivalent Series Resistor Rs	Fundamental @ 16 MHz			60	Ω
C_M	Motional Capacitance		5		9	fF
C_S	Shunt Capacitance				7	pF

34.5.4 PLLA Characteristics

Table 34-10. Phase Lock Loop A Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
F_{OUT}	Output Frequency	Field OUT of CKGR_PLL is 00	80		200	MHz
		Field OUT of CKGR_PLL is 10	190		240	MHz
F_{IN}	Input Frequency		1		32	MHz
I_{PLL}	Current Consumption	Active mode (@240 MHz)			3	mA
		Standby mode			1	µA

Note: 1. Startup time depends on PLL RC filter. A calculation tool is provided by Atmel.

34.5.5 PLLB Characteristics

Table 34-11. Phase Lock Loop B Characteristics

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
F_{OUT}	Output Frequency		50		150	MHz
F_{IN}	Input Frequency		1		32	MHz
I_{PLL}	Current Consumption	Active mode (@150 MHz)			2.5	mA
		Standby mode			TBD	µA

Note: 1. PLLB feature an embedded PLL RC filter.

34.6 USB Transceiver Characteristics

34.6.1 Electrical Characteristics

Table 34-12. USB Transceiver 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 kΩ tied to 3.6V	0.0		0.3	V
V_{OH}	High Level Output	Measured with R_L of 14.25 kΩ tied to GND	2.8		3.6	V
V_{CRS}	Output Signal Crossover Voltage	Measure conditions described in Figure 34-1	1.3		2.0	V
Pull-up Resistor						
R_{PUI}	Bus Pull-up Resistor on Upstream Port (idle bus)		0.900		1.575	kOhm
R_{PUA}	Bus Pull-up Resistor on Upstream Port (upstream port receiving)		1.425		3.090	kOhm

34.6.2 Switching Characteristics

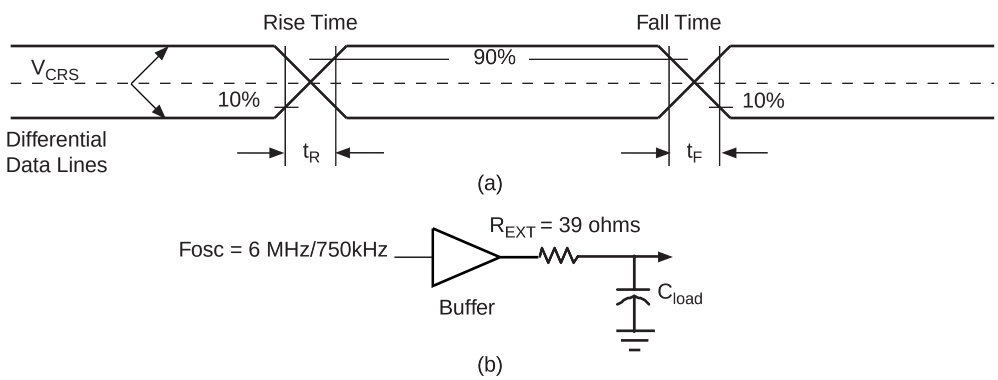
Table 34-13. In Full Speed

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 34-14. 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 34-1. USB Data Signal Rise and Fall Times



34.7 EBI Timings

The following characteristics are applicable to the operating temperature range: T_A = -40°C to 85°C and power supply worst case, unless otherwise specified.

These timings are given for operating temperature range: T_A = -40°C to 85°C and VDDCORE = 1.08V to 1.32V.

First column for VDDIOM in 1.8V supply range (1.65V to 1.95V) and 30 pF load capacitance.

Second column for VDDIOM in 3.3V supply range (3.0V to 3.6V) and 50 pF load capacitance.

Table 34-15. SMC Read Signals with Hold Settings

Symbol	Parameter	Min		Units
		1.8V Supply	3.3V Supply	
NRD Controlled (READ_MODE = 1)				
SMC_1	Data Setup before NRD High	TBD	+5.0	ns
SMC_2	Data Hold after NRD High	TBD	-1.9	ns
SMC_3	NRD High to NBS0/A0 Change [(1)]	TBD	nrd hold length * t_{CPMCK} + 0.2	ns
SMC_4	NRD High to NBS1 Change [(1)]	TBD	nrd hold length * t_{CPMCK} + 0.2	ns
SMC_5	NRD High to NBS2/A1 Change [(1)]	TBD	nrd hold length * t_{CPMCK} + 0.2	ns
SMC_6	NRD High to NBS3 Change [(1)]	TBD	nrd hold length * t_{CPMCK} + 0.2	ns
SMC_7	NRD High to A2 - A25 Change [(1)]	TBD	nrd hold length * t_{CPMCK} + 0.3	ns
SMC_8	NRD High to NCS Inactive [(1)]	TBD	(nrd hold length - ncs rd hold length) * t_{CPMCK} - 0.2	ns
SMC_9	NRD Pulse Width	TBD	nrd pulse length * t_{CPMCK} - 0.5	ns
NCS Controlled (READ_MODE = 0)				
SMC_{10}	Data Setup before NCS High	TBD	+4.8	ns
SMC_{11}	Data Hold after NCS High	TBD	-1.9	ns
SMC_{12}	NCS High to NBS0/A0 Change [(1)]	TBD	ncs rd hold length * t_{CPMCK} + 0.4	ns
SMC_{13}	NCS High to NBS1 Change [(1)]	TBD	ncs rd hold length * t_{CPMCK} + 0.4	ns
SMC_{14}	NCS High to NBS2/A1 Change [(1)]	TBD	ncs rd hold length * t_{CPMCK} + 0.4	ns
SMC_{15}	NCS High to NBS3 Change[(1)]	TBD	ncs rd hold length * t_{CPMCK} + 0.4	ns
SMC_{16}	NCS High to A2 - A25 Change [(1)]	TBD	ncs rd hold length * t_{CPMCK} + 0.3	ns
SMC_{17}	NCS High to NRD Inactive [(1)]	TBD	(ncs rd hold length - nrd hold length)* t_{CPMCK} + 0.0	ns
SMC_{18}	NCS Pulse Width	TBD	ncs rd pulse length * t_{CPMCK} - 0.5	ns

Notes: 1. hold length = total cycle duration - setup duration - pulse duration. “hold length” is for “ncs rd hold length” or “nrd hold length”.

Table 34-16. SMC Read Signals with No Hold Settings

Symbol	Parameter	Min		Units
		1.8V Supply	3.3V Supply	
NRD Controlled (READ_MODE = 1)				
SMC_{19}	Data Setup before NRD High	TBD	5.0	ns
SMC_{20}	Data Hold after NRD High	TBD	-1.9	ns
NCS Controlled (READ_MODE = 0)				
SMC_{21}	Data Setup before NCS High	TBD	4.8	ns
SMC_{22}	Data Hold after NCS High	TBD	-1.8	ns

Table 34-17. SMC Write Signals with Hold Settings

Symbol	Parameter	Min		Units
		1.8V Supply	3.3V Supply	
NWE Controlled (WRITE_MODE = 1)				
SMC_{23}	Data Out Valid before NWE High	TBD	(nwe pulse length) * t_{CPMCK} - 0.2	ns
SMC_{24}	Data Out Valid after NWE High [(1)]	TBD	nwe hold length * t_{CPMCK} - 0.5	ns
SMC_{25}	NWE High to NBS0/A0 Change [(1)]	TBD	nwe hold length * t_{CPMCK} - 0.0	ns
SMC_{26}	NWE High to NBS1 Change [(1)]	TBD	nwe hold length * t_{CPMCK} - 0.0	ns
SMC_{29}	NWE High to NBS2/A1 Change [(1)]	TBD	nwe hold length * t_{CPMCK} - 0.0	ns
SMC_{30}	NWE High to NBS3 Change [(1)]	TBD	nwe hold length * t_{CPMCK} - 0.0	ns
SMC_{31}	NWE High to A2 - A25 Change [(1)]	TBD	nwe hold length * t_{CPMCK} - 0.0	ns
SMC_{32}	NWE High to NCS Inactive[(1)]	TBD	(nwe hold length - ncs wr hold length)* t_{CPMCK} - 0.5	ns
SMC_{33}	NWE Pulse Width	TBD	nwe pulse length * t_{CPMCK} + 0.2	ns
NCS Controlled (WRITE_MODE = 0)				
SMC_{34}	Data Out Valid before NCS High	TBD	(ncs wr pulse length)* t_{CPMCK} - 0.4	ns
SMC_{35}	Data Out Valid after NCS High [(1)]	TBD	ncs wr hold length * t_{CPMCK} - 0.3	ns
SMC_{36}	NCS High to NWE Inactive [(1)]	TBD	(ncs wr hold length - nwe hold length)* t_{CPMCK} + 0.2	ns

Note: 1. hold length = total cycle duration - setup duration - pulse duration. “hold length” is for “ncs wr hold length” or “nwe hold length”.

Table 34-18. SMC Write Signals with No Hold Settings (NWE Controlled only)

Symbol	Parameter	Min		Units
		1.8V Supply	3.3V Supply	
SMC_{37}	NWE Rising to A2-A25 Valid	TBD	0.1	ns
SMC_{38}	NWE Rising to NBS0/A0 Valid	TBD	0.0	ns
SMC_{39}	NWE Rising to NBS1 Change	TBD	0.0	ns
SMC_{40}	NWE Rising to A1/NBS2 Change	TBD	0.0	ns
SMC_{41}	NWE Rising to NBS3 Change	TBD	0.0	ns
SMC_{42}	NWE Rising to NCS Rising	TBD	3.2	ns
SMC_{43}	Data Out Valid before NWE Rising	TBD	(nwe pulse length)* t_{CPMCK} - 0.9	ns
SMC_{44}	Data Out Valid after NWE Rising	TBD	3.0	ns
SMC_{45}	NWE Pulse Width	TBD	nwe pulse length * t_{CPMCK} - 0.2	ns

Figure 34-2. SMC Signals for NCS Controlled Accesses

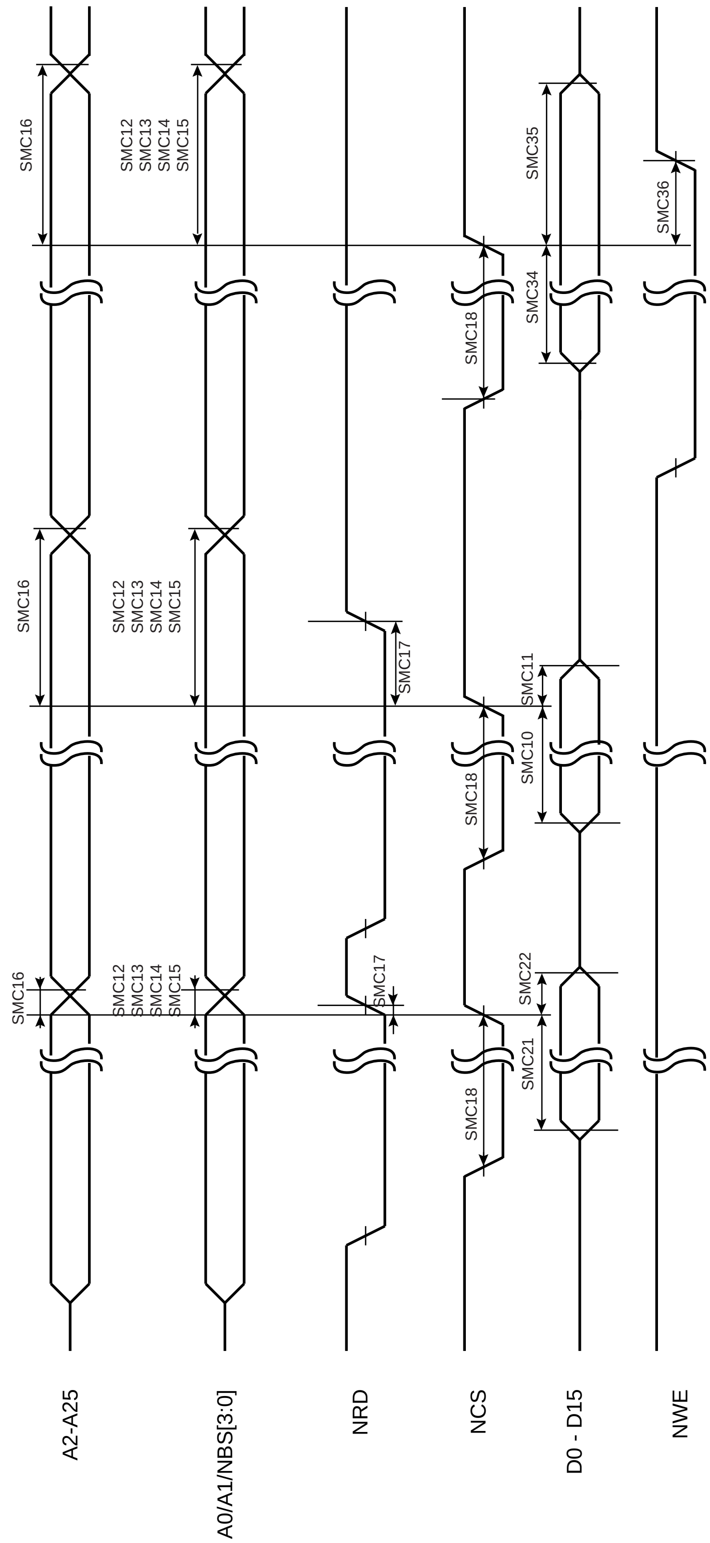
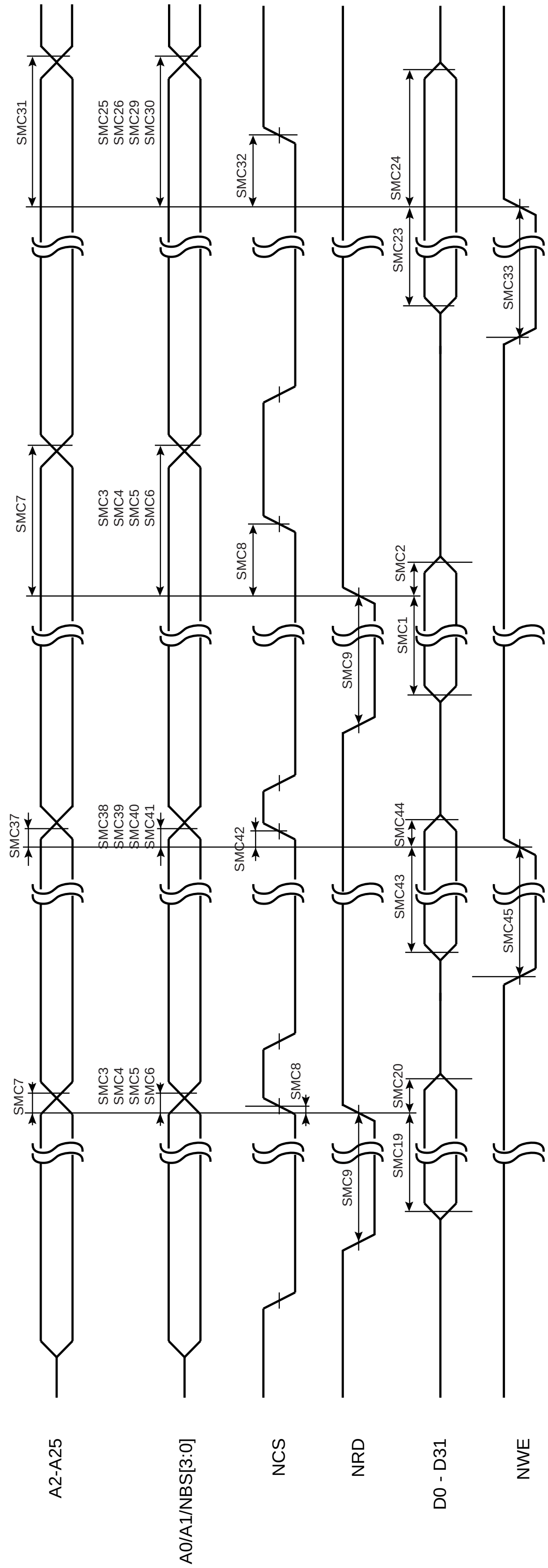


Figure 34-3. SMC Signals for NRD and NWR Controlled Accesses



34.7.1 SDRAMC Signals

These timings are given for a 10 pF load on SDCK and 50 pF on the data bus.

Table 34-19. SDRAMC Clock Signal

Symbol	Parameter	Max		Units
		1.8V Supply	3.3V Supply	
$1/(t_{CPSDCK})$	SDRAM Controller Clock Frequency	100	100	MHz

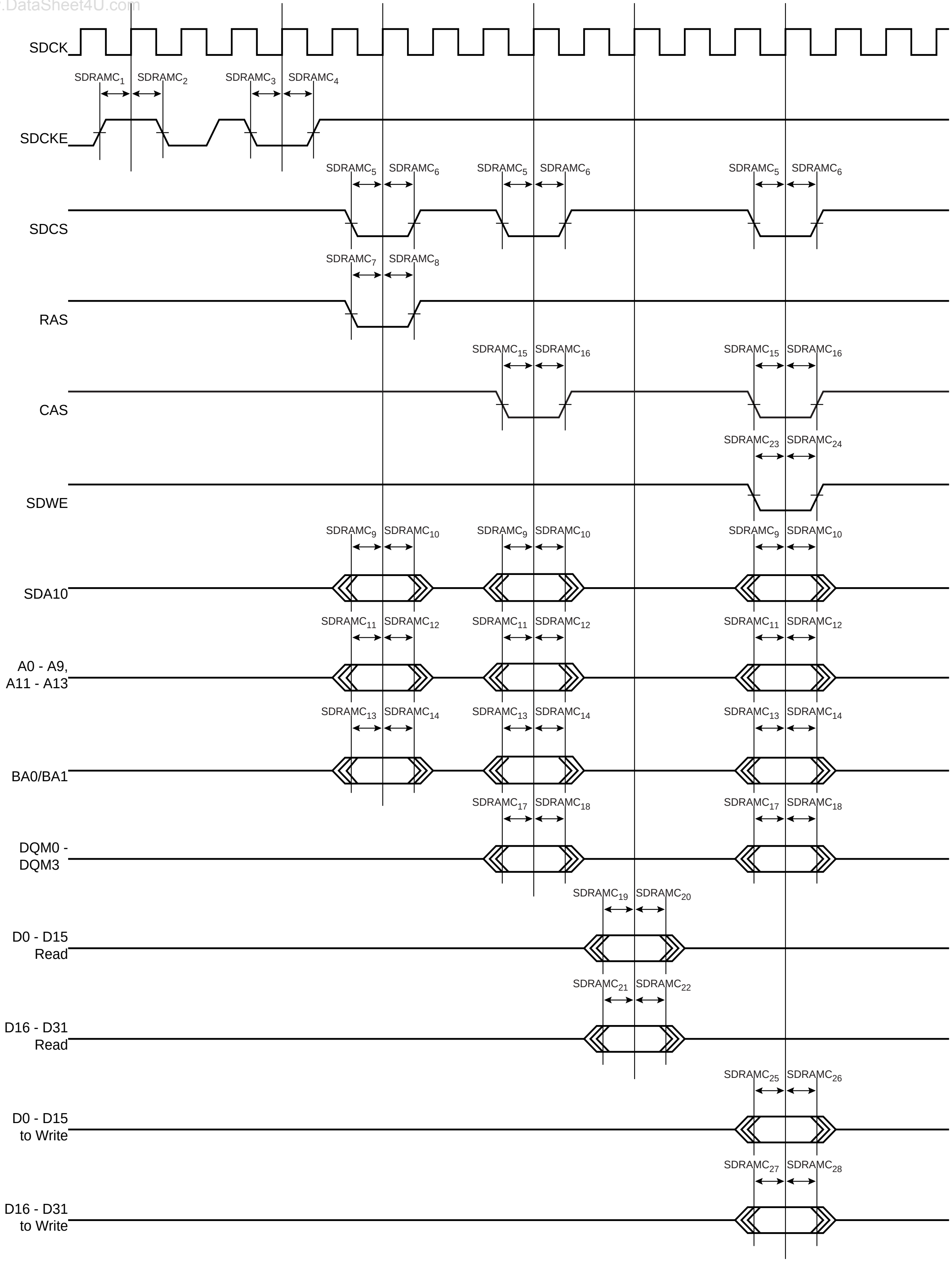
Table 34-20. SDRAMC Signals

Symbol	Parameter	Min		Units
		1.8V Supply	3.3V Supply	
$SDRAMC_1$	SDCKE High before SDCK Rising Edge	TBD	4.7	ns
$SDRAMC_2$	SDCKE Low after SDCK Rising Edge	TBD	6.8	ns
$SDRAMC_3$	SDCKE Low before SDCK Rising Edge	TBD	4.5	ns
$SDRAMC_4$	SDCKE High after SDCK Rising Edge	TBD	n.a.	ns
$SDRAMC_5$	SDCS Low before SDCK Rising Edge	TBD	4.8	ns
$SDRAMC_6$	SDCS High after SDCK Rising Edge	TBD	6.5	ns
$SDRAMC_7$	RAS Low before SDCK Rising Edge	TBD	4,7	ns
$SDRAMC_8$	RAS High after SDCK Rising Edge	TBD	6.4	ns
$SDRAMC_9$	SD_A10 Change before SDCK Rising Edge	TBD	5.2	ns
$SDRAMC_{10}$	SD_A10 Change after SDCK Rising Edge	TBD	6.6	ns
$SDRAMC_{11}$	Address Change before SDCK Rising Edge	TBD	5.3	ns
$SDRAMC_{12}$	Address Change after SDCK Rising Edge	TBD	6.7	ns
$SDRAMC_{13}$	Bank Change before SDCK Rising Edge	TBD	5.2	ns
$SDRAMC_{14}$	Bank Change after SDCK Rising Edge	TBD	6.5	ns
$SDRAMC_{15}$	CAS Low before SDCK Rising Edge	TBD	4.9	ns
$SDRAMC_{16}$	CAS High after SDCK Rising Edge	TBD	6.5	ns
$SDRAMC_{17}$	DQM Change before SDCK Rising Edge	TBD	5.0	ns
$SDRAMC_{18}$	DQM Change after SDCK Rising Edge	TBD	6.3	ns
$SDRAMC_{19}$	D0-D15 in Setup before SDCK Rising Edge	TBD	0.5	ns
$SDRAMC_{20}$	D0-D15 in Hold after SDCK Rising Edge	TBD	0.1	ns
$SDRAMC_{21}$	D16-D31 in Setup before SDCK Rising Edge	TBD	0.5	ns
$SDRAMC_{22}$	D16-D31 in Hold after SDCK Rising Edge	TBD	0.1	ns
$SDRAMC_{23}$	SDWE Low before SDCK Rising Edge	TBD	4.4	ns
$SDRAMC_{24}$	SDWE High after SDCK Rising Edge	TBD	6.6	ns
$SDRAMC_{25}$	D0-D15 Out Valid before SDCK Rising Edge	TBD	6.0	ns

Table 34-20. SDRAMC Signals

Symbol	Parameter	Min 1.8V Supply	Min 3.3V Supply	Units
$SDRAMC_{26}$	D0-D15 Out Valid after SDCK Rising Edge	TBD	5.8	ns
$SDRAMC_{27}$	D16-D31 Out Valid before SDCK Rising Edge	TBD	6.0	ns
$SDRAMC_{28}$	D16-D31 Out Valid after SDCK Rising Edge	TBD	5.8	ns

Figure 34-4. SDRAMC Signals Relative to SDCK



35. Mechanical Characteristics

35.1 Thermal Considerations

35.1.1 Thermal Data

Table 35-1 summarizes the thermal resistance data depending on the package.

Table 35-1. Thermal Resistance Data

Symbol	Parameter	Condition	Package	Typ	Unit
θ_{JA}	Junction-to-ambient thermal resistance	Still Air	CABGA324	TBD	°C/W
θ_{JC}	Junction-to-case thermal resistance		CABGA324	TBD	

35.1.2 Junction Temperature

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

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

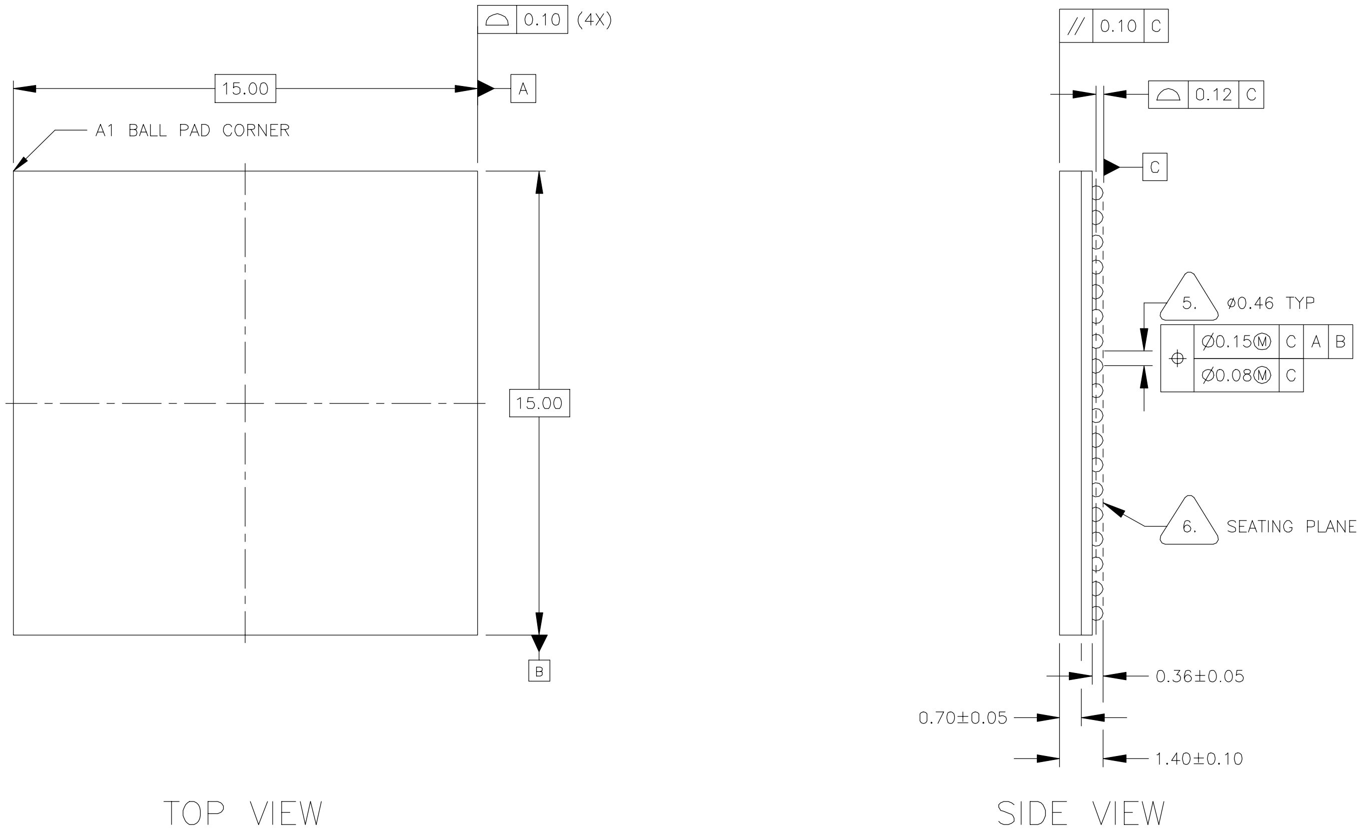
where:

- θ_{JA} = package thermal resistance, Junction-to-ambient (°C/W), provided in Table 35-1 on page 732.
- θ_{JC} = package thermal resistance, Junction-to-case thermal resistance (°C/W), provided in Table 35-1 on page 732.
- $\theta_{HEAT\ SINK}$ = 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 716.
- T_A = ambient temperature (°C).

From the first equation, the user can derive the estimated lifetime of the chip and decide whether 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.

35.2 Package Drawings

Figure 35-1. 324-ball CABGA Package Drawing (dimensions in mm)



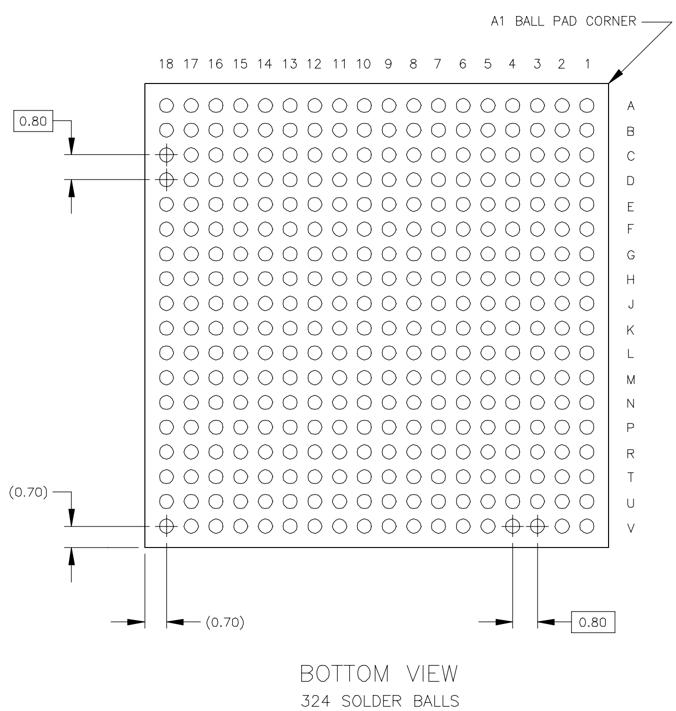


Table 35-2. Soldering Information

Ball Land	0.50 mm ± 0.3
Ball Diameter	0.40 mm ± 0.3
Solder Mask Opening	0.35 mm ± 0.3

Table 35-3. Device and 324-ball CABGA Package Maximum Weight

482.5	mg

Table 35-4. 324-ball CABGA Package Characteristics

Moisture Sensitivity Level	3

Table 35-5. Package Reference

JEDEC Drawing Reference	MO-205
JESD97 Classification	e1

35.3 Soldering Profile

Table 35-6 gives the recommended soldering profile.

Table 35-6. Soldering Profile

Profile Feature	Green Package
Average Ramp-up Rate (217°C to Peak)	1°C/sec. max.
Preheat Temperature	150°C to 200°C
Time Maintained Above 217°C	36 sec
Time within 5°C of Actual Peak Temperature	20 sec
Peak Temperature Range	240 +0 °C
Ramp-down Rate	3°C/sec. max.
Time 25°C to Peak Temperature	8 min. max.

Maximum three reflow passes are allowed per each component.

36. Ordering Guide

Table 36-1. Ordering Information

Part Number	Temp. Range	Speed Grade (Max)	Operating Voltage	Package	Notes	Status
AT572D940HF	0°C to 70°C	160 MHz	3.3V (I/O) 1.1V (core)	CA324BGA (RoHS)	Full Peripheral Set	Sampling
AT572D940HF-CL	0°C to 70°C	160 MHz	1.8V-2.5V-3.3V (I/O) 1.2V (core)	CA324BGA (RoHS)	Reduced Periperal Set [1]	Contact: diopsis@atmel.com
AT572D940HF-CJ	-40°C to 85°C	200 MHz	1.8V-2.5V-3.3V (I/O) 1.2V (core)	CA324BGA (RoHS)	Full Peripheral Set	Contact: diopsis@atmel.com

1. Some peripherals are not accessible by the user in this low-cost version. Reduced Peripheral Set = Full Peripheral Set - 2 CANs -3 SSCs - 1 SPI - 1 TWI - 2 USARTs. Consequently the related PIO lines can be used only as SW controlled PIO lines (not linked to any peripherals).

37. Revision History

Doc. Rev.	Date	Comments
7010A	07/08	• Initial document release

Table of Contents

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel 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

Product Contact

Web Site
www.atmel.com

Technical Support
diopsis@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature