Freescale Semiconductor

MCF54455RM
Rev. 6.1, 03/2012

MCF54455 Reference Manual

by: Microcontroller Solutions Group

This is the MCF54455 Reference Manual set consisting of the following files:

- MCF54455 Reference Manual Errata, Rev 1
- MCF54455 Reference Manual, Rev 6

MCF54455RMAD
Rev. 1, 11/2011

MCF54455 Reference Manual Errata

by: Microcontroller Solutions Group

This errata document describes corrections to the *MCF54455 Reference Manual*, order number MC54455RM. For convenience, the addenda items are grouped by revision. Please check our website at http://www.freescale.com for the latest updates.

The current available version of the *MCF54455 Reference Manual* is Revision 6.

Table of Contents

1 Errata for Revision 6

Table 1. MCF54455 Reference Manual Rev 6 Errata

Location	Description
Section 16.2, “External Signal Description”/Table 16-2/Page 16-11	Add pin N7 to the VSS pin list for the 360 TEPBGA.

2 Revision History

Table 2 provides a revision history for this document.

Table 2. Revision History Table

Rev. Number	Substantive Changes	Date of Release
1.0	Initial release. Correct errors in section 16.2, “External Signal Description”.	11/2011

THIS PAGE IS INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd. Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

MCF54455 Reference Manual

Devices Supported:

MCF54450
MCF54451
MCF54452
MCF54453
MCF54454
MCF54455

Document Number: MCF54455RM
Rev. 6
5/2011

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

MCF54455RM
Rev. 6
5/2011

Table of Contents

Chapter 1
Overview

Chapter 2
Signal Descriptions

Chapter 3
ColdFire Core

Chapter 4
Memory Management Unit (MMU)

Chapter 5
Enhanced Multiply-Accumulate Unit (EMAC)

Chapter 6
Cache

Chapter 7
Static RAM (SRAM)

Chapter 8
Clock Module

Chapter 9
Power Management

Chapter 10
Universal Serial Bus Interface – On-The-Go Module

Chapter 11
Chip Configuration Module (CCM)

Chapter 12
Serial Boot Facility (SBF)

Chapter 13
Reset Controller Module

Chapter 14
System Control Module (SCM)

Chapter 15
Crossbar Switch (XBS)

Chapter 16
Pin Multiplexing and Control

Chapter 17
Interrupt Controller Modules

Chapter 18
Edge Port Module (EPORT)

Chapter 19
Enhanced Direct Memory Access (eDMA)

Chapter 20

FlexBus

Chapter 21
SDRAM Controller (SDRAMC)

Chapter 22
PCI Bus Controller

Chapter 23
Advanced Technology Attachment (ATA)

Chapter 24
Cryptographic Acceleration Unit (CAU)

Chapter 25
Random Number Generator (RNG)

Chapter 26
Fast Ethernet Controllers (FEC0 and FEC1)

Chapter 27
Synchronous Serial Interface (SSI)

Chapter 28
Real-Time Clock

Chapter 29
Programmable Interrupt Timers (PIT0–PIT3)

Chapter 30
DMA Timers (DTIM0–DTIM3)

Chapter 31
DMA Serial Peripheral Interface (DSPI)

Chapter 32
UART Modules

Chapter 33
I^{2}C Interface

Chapter 34
Debug Module

Chapter 35
IEEE 1149.1 Test Access Port (JTAG)

Appendix A
Revision History

About This Book

The primary objective of this reference manual is to define the processor for software and hardware developers. The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, the reader must use the most recent version of the documentation.

To locate any published errata or updates for this document, refer to the world-wide web at http://www.freescale.com/coldfire.

Portions of Chapter 23, "Universal Serial Bus Interface – Host Module," and Chapter 10, "Universal Serial Bus Interface – On-The-Go Module,"relating to the EHCI specification are Copyright © Intel Corporation 1999-2001. The EHCI specification is provided "As Is" with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in the EHCI specification. Intel may make changes to the EHCI specifications at any time, without notice.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with this ColdFire processor. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the ColdFire® architecture.

Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about ColdFire architecture.

General Information

Useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual* (MCF5200PRM/AD)
- *Using Microprocessors and Microcomputers: The Motorola Family,* William C. Wray, Ross Bannatyne, Joseph D. Greenfield
- *Computer Architecture: A Quantitative Approach*, Second Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, David A. Patterson and John L. Hennessy.

ColdFire Documentation

ColdFire documentation is available from the sources listed on the back cover of this manual, as well as our web site, http://www.freescale.com/coldfire.

- Reference manuals — These books provide details about individual ColdFire implementations and are intended to be used in conjunction with the *ColdFire Programmers Reference Manual.*
- Data sheets — Data sheets provide specific data regarding pin-out diagrams, bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product briefs — Each device has a product brief that provides an overview of its features. This document is roughly equivalent to the overview (Chapter 1) of an device's reference manual.
- Application notes — These short documents address specific design issues useful to programmers and engineers working with Freescale Semiconductor processors.

Additional literature is published as new processors become available. For a current list of ColdFire documentation, refer to http://www.freescale.com/coldfire.

Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
`mnemonics`	In code and tables, instruction mnemonics are shown in lowercase.
italics	Italics indicate variable command parameters. Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields, or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base address register.
nibble	A 4-bit data unit
byte	An 8-bit data unit
word	A 16-bit data unit[1]
longword	A 32-bit data unit
x	In some contexts, such as signal encodings, x indicates a don't care.
n	Used to express an undefined numerical value
~	NOT logical operator
&	AND logical operator
|	OR logical operator

[1] The only exceptions to this appear in the discussion of serial communication modules that support variable-length data transmission units. To simplify the discussion these units are referred to as words regardless of length.

||	Field concatenation operator
$\overline{\text{OVERBAR}}$	An overbar indicates that a signal is active-low.

Register Figure Conventions

This document uses the following conventions for the register reset values:

—	Undefined at reset.
u	Unaffected by reset.
[*signal_name*]	Reset value is determined by the polarity of the indicated signal.

The following register fields are used:

R	0	Indicates a reserved bit field in a memory-mapped register. These bits are always read as zeros.
W	▨	
R	1	Indicates a reserved bit field in a memory-mapped register. These bits are always read as ones.
W	▨	
R	FIELDNAME	Indicates a read/write bit.
W		
R	FIELDNAME	Indicates a read-only bit field in a memory-mapped register.
W	▨	
R	▨	Indicates a write-only bit field in a memory-mapped register.
W	FIELDNAME	
R	FIELDNAME	Write 1 to clear: indicates that writing a 1 to this bit field clears it.
W	w1c	
R	0	Indicates a self-clearing bit.
W	FIELDNAME	

Chapter 1
Overview

The MCF5445*x* devices are a family of highly-integrated 32-bit microprocessors based on the Version 4 ColdFire microarchitecture. This product line is well suited for secure networked applications in factory automation, process control, and motion control. The rich feature set and flexibility make it attractive to many different applications in consumer and industrial markets.

All MCF5445*x* devices contain a Version 4 ColdFire core, 32-Kbyte internal SRAM, USB On-the-Go controllers, a 2-bank DDR/DDR2/mobile-DDR SDRAM controller, a 16-channel DMA controller, a serial boot facility, an SSI interface, and other serial interfaces. Optional peripherals include a PCI bus controller, ATA controller, Fast Ethernet controllers, and an encryption coprocessor.

1.1 MCF5445*x* Family Comparison

The following table compares the various device derivatives available within the MCF5445*x* family.

Table 1-1. MCF5445*x* Family Configurations

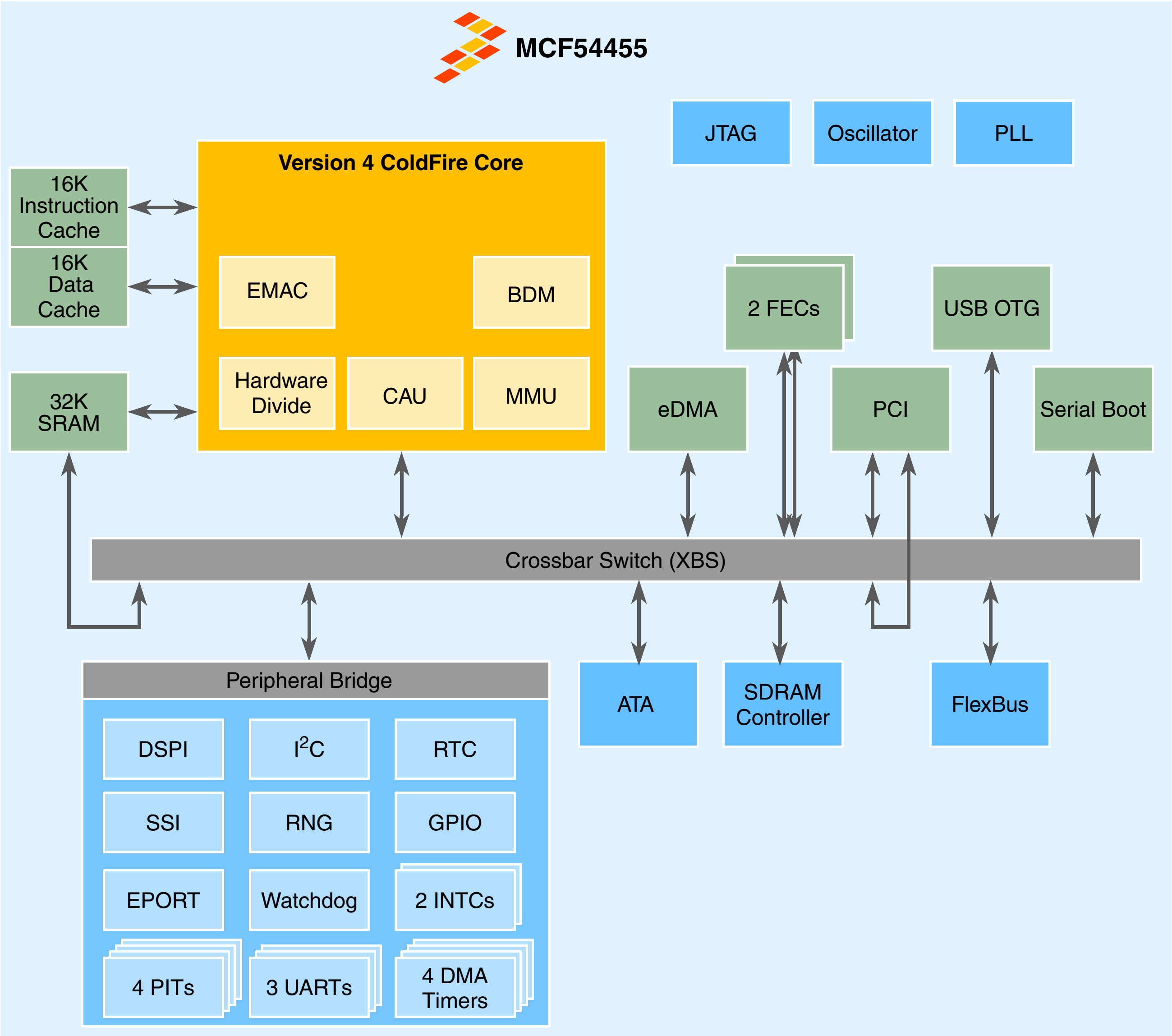
Module	MCF54450	MCF54451	MCF54452	MCF54453	MCF54454	MCF54455
ColdFire Version 4 Core with EMAC (Enhanced Multiply-Accumulate Unit)	•	•	•	•	•	•
Core (System) Clock	up to 240 MHz		up to 266 MHz			
Peripheral Bus Clock (Core clock ÷ 2)	up to 120 MHz		up to 133 MHz			
External Bus Clock (Core clock ÷ 4)	up to 60 MHz		up to 66 MHz			
Performance (Dhrystone/2.1 MIPS)	up to 370		up to 410			
Independent Data/Instruction Cache	16 Kbytes each					
Static RAM (SRAM)	32 Kbytes					
PCI Controller	—	—	•	•	•	•
Cryptography Acceleration Unit (CAU)	—	•	—	•	—	•
ATA Controller	—	—	—	—	•	•
DDR SDRAM Controller	•	•	•	•	•	•
FlexBus External Interface	•	•	•	•	•	•
USB 2.0 On-the-Go	•	•	•	•	•	•
UTMI+ Low Pin Interface (ULPI)	•	•	•	•	•	•
Synchronous Serial Interface (SSI)	•	•	•	•	•	•

Table 1-1. MCF5445*x* Family Configurations (continued)

Module	MCF54450	MCF54451	MCF54452	MCF54453	MCF54454	MCF54455
Fast Ethernet Controller (FEC)	1	1	2	2	2	2
UARTs	3	3	3	3	3	3
I^{2}C	•	•	•	•	•	•
DSPI	•	•	•	•	•	•
Real Time Clock	•	•	•	•	•	•
32-bit DMA Timers	4	4	4	4	4	4
Watchdog Timer (WDT)	•	•	•	•	•	•
Periodic Interrupt Timers (PIT)	4	4	4	4	4	4
Edge Port Module (EPORT)	•	•	•	•	•	•
Interrupt Controllers (INTC)	2	2	2	2	2	2
16-channel Direct Memory Access (DMA)	•	•	•	•	•	•
General Purpose I/O (GPIO)	•	•	•	•	•	•
JTAG - IEEE® 1149.1 Test Access Port	•	•	•	•	•	•
Package	256 MAPBGA		360 TEPBGA			

1.2 Block Diagram

Figure 1-1 shows a top-level block diagram of the MCF54455 superset device.



LEGEND

ATA	– Advanced Technology Attachment Controller	**INTC**	– Interrupt controller
BDM	– Background debug module	**JTAG**	– Joint Test Action Group interface
CAU	– Cryptography acceleration unit	**MMU**	– Memory management unit
DSPI	– DMA serial peripheral interface	**PCI**	– Peripheral Component Interconnect
eDMA	– Enhanced direct memory access	**PIT**	– Programmable interrupt timers
EMAC	– Enchance multiply-accumulate unit	**PLL**	– Phase locked loop module
EPORT	– Edge port module	**RNG**	– Random Number Generator
FEC	– Fast Ethernet controller	**RTC**	– Real time clock
GPIO	– General Purpose Input/Output	**SSI**	– Synchronous Serial Interface
I^{2}C	– Inter-Intergrated Circuit	**USB OTG**	– Universal Serial Bus On-the-Go controller

Figure 1-1. MCF54455 Block Diagram

1.3 Operating Parameters

- 0ºC to 70ºC and –40ºC to 85ºC junction temperature devices are available
- 1.5V Core, 3.3V I/O, 1.8V/2.5V/3.3V external memory bus

1.4 Packages

Depending on device, the MCF5445*x* family is available in the following packages:

- 256-pin molded array process ball grid array (MAPBGA)
- 360-pin plastic ball grid array (TEPBGA)

1.5 Chip Level Features

- Version 4 ColdFire core with MMU and EMAC
- Up to 410 Dhrystone 2.1 MIPS @ 266 MHz
- 16 Kbytes instruction cache and 16 Kbytes data cache
- 32 Kbytes internal SRAM
- Support for booting from SPI-compatible flash, EEPROM, and FRAM devices
- Crossbar switch technology (XBS) for concurrent access to peripherals or RAM from multiple bus masters
- 16 channel DMA controller
- 16-bit 133MHz DDR/mobile-DDR/DDR2 Controller
- USB 2.0 On-the-Go controller with ULPI support
- 32-bit PCI controller at 66 MHz
- ATA/ATAPI controller
- 2 10/100 Ethernet MACs
- Coprocessor for acceleration of the DES, 3DES, AES, MD5, and SHA-1 algorithms
- Random number generator
- Synchronous serial interface (SSI)
- 4 periodic interrupt timers (PIT)
- 4 32-bit timers with DMA support
- DMA supported serial peripheral interface (DSPI)
- 3 UARTs
- I^{2}C bus interface

1.6 Module-by-Module Feature List

The following is a brief summary of the functional blocks in the MCF54455 superset device. For more details refer to the *MCF54455 ColdFire Microprocessor Reference Manual* (MCF54455RM).

1.6.1 Version 4 ColdFire variable-length RISC processor

- Static operation
- 32-bit address and data path on-chip
- Maximum 266 MHz processor core, 133 MHz internal peripheral, and 66 MHz external FlexBus frequency
- Sixteen total general-purpose 32-bit registers data and address
- Enhanced multiply-accumulate unit (EMAC) for DSP and fast multiply operations
- Hardware divide execution unit supporting various 32-bit operations
- Implements the ColdFire Instruction Set Architecture, ISA_C
- Cryptography acceleration unit (CAU)
  - DES and AES block cipher engines
  - MD5, SHA-1, and HMAC hash accelerator

1.6.2 On-chip Memories

- 32 Kbyte dual-ported SRAM on CPU internal bus
  - Accessible to non-core bus masters (e.g. FEC, DMA, USB OTG, and PCI controllers) via the crossbar switch
- Non-blocking, independent 16 Kbyte data and instruction caches organized as 4-way set associative with 16 bytes per cache line and 1024 cache lines, supporting copy-back and write-through modes of operation

1.6.3 Phase Locked Loop (PLL)

- 16–40 MHz reference crystal
- Loss-of-lock detection

1.6.4 Power Management

- Fully static operation with processor sleep and whole chip stop modes
- Very rapid response to interrupts from the low-power sleep mode (wake-up feature)
- Peripheral power management register to enable/disable clocks to most modules
- Software controlled disable of external clock input for low power consumption

1.6.5 Chip Configuration Module (CCM)

- System configuration during reset
- Bus monitor, abort monitor
- Configurable output pad drive strength control
- Unique part identification and part revision numbers
- Serial boot capability

- — Supports SPI-compatible EEPROM, flash, and FRAM
- — Configurable boot clock frequency

1.6.6 Reset Controller

- Separate reset in and reset out signals
- Six sources of reset: power-on reset (POR), external, software, watchdog timer, loss of lock, JTAG instruction
- Status flag indication of source of last reset

1.6.7 System Control Module

- Access control registers
- Core watchdog timer with a 2^n (where n = 8–31) clock cycle selectable timeout period
- Core fault reporting

1.6.8 Crossbar Switch

- Concurrent access from different masters to different slaves
- Slave arbitration attributes configured on a slave by slave basis
- Fixed or round-robin arbitration

1.6.9 Peripheral Component Interconnect (PCI) Bus

- Compatible with PCI 2.2 specification
- Supports up to 4 external PCI masters
- 32-bit target and intiator operation
- 33–66 MHz operation with PCI bus to internal bus divider ratios of 1:1, 1:2, 1:3, 2:3, and 1:4

1.6.10 Universal Serial Bus (USB) 2.0 On-The-Go (OTG) Controller

- Support for full speed (FS) and low speed (LS) via a serial interface or on-chip FS/LS transceiver
- Optional UTMI+ Low Pin Count Interface (ULPI) on some packages to support high speed (HS) transfers
- Uses 60 MHz reference clock based off of the system clock or from an external pin

1.6.11 DDR SDRAM Controller

- Supports a glueless interface to DDR, DDR2, and mobile/low-power DDR SDRAM devices
- Support for 16-bit fixed memory port width
- 16-byte critical word first burst transfer
- Up to 14 lines of row address, up to 11 column address lines (16-bit bus), 2 bits of bank address, and two pinned-out chip selects. The maximum row bits plus column bits equals 25.

- Supports up to 512 MByte of memory; minimum memory configuration of 8 MByte
- Supports page mode to maximize the data rate
- Supports sleep mode and self-refresh mode

1.6.12 FlexBus (External Interface)

- Glueless connections to 16-, and 32-bit external memory devices (SRAM, flash, ROM, etc.)
- Support for independent primary and secondary wait states per chip select
- Programmable address setup and hold time with respect to chip-select assertion, per transfer direction
- Glueless interface to SRAM devices with or without byte strobe inputs
- Programmable wait state generator
- 32-bit external bidirectional data bus and 24-bit address bus
- Up to four chip selects available
- Byte/write enables (byte strobes)
- Ability to boot from external memories that are 8, 16, or 32 bits wide

1.6.13 Synchronous Serial Interface (SSI)

- Supports shared (synchronous) transmit and receive sections
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with as many as 32 time slots
- Gated clock mode operation requiring no frame sync
- Programmable data interface modes such as I^2S, LSB aligned, and MSB aligned
- Programmable word length up to 24 bits
- AC97 support

1.6.14 ATA Controller

- Compliant with ATA-6 specification
- Supports PIO modes 0, 1, 2, 3 and 4
- Supports multiword DMA modes 0, 1 and 2
- Supports ultra DMA modes 0, 1, 2, 3 and 4 with an internal bus clock of at least 50 Mhz
- Supports ultra DMA mode 5 with an internal bus clock of at least 80 Mhz
- 128 byte FIFO part of interface
- FIFO receive alarm, FIFO transmit alarm and FIFO end of transmission alarm to DMA unit
- Zero-wait cycles transfer between DMA bus and FIFO allows fast FIFO reading/writing

1.6.15 Fast Ethernet Media Access Controller (FEC MAC)

- 10/100 BaseT/TX capability, half duplex or full duplex

- On-chip transmit and receive FIFOs
- Built-in dedicated DMA controller
- Memory-based flexible descriptor rings
- Media independent interface (MII) to external transceiver (PHY)
- Separate RMII gasket to interface with RMII-compatible PHY

1.6.16 Random Number Generator (RNG)

- FIPS-140 compliant for randomness and non-determinism

1.6.17 Real Time Clock

- Full clock: days, hours, minutes, seconds
- Minute countdown timer with interrupt
- Programmable daily alarm with interrupt
- Sampling timer with interrupt
- Once-per-day, once-per-hour, once-per-minute, and once-per-second interrupts
- Operation determined by reference input oscillator clock frequency and value programmed into user-accessible registers
- Ability to wake the processor from low-power modes (wait, doze, and stop) via the RTC interrupts

1.6.18 Software Watchdog Timer

- 16-bit down-counter which resets the device if not serviced

1.6.19 Programmable Interrupt Timers (PIT)

- Four programmable interrupt timers each with a 16-bit counter
- Configurable as a down counter or free-running counter

1.6.20 DMA Timers

- Four 32-bit timers with DMA and interrupt request trigger capability
- Input capture and reference compare modes

1.6.21 DMA Serial Peripheral Interface (DSPI)

- Full-duplex, three-wire synchronous transfer
- Up to five chip selects available
- Master and slave modes with programmable master bit-rates
- Up to 16 pre-programmed transfers

1.6.22 Universal Asynchronous Receiver Transmitters (UARTs)

- 16-bit divider for clock generation
- Interrupt control logic
- DMA support with separate transmit and receive requests
- Programmable clock-rate generator
- Data formats can be 5, 6, 7 or 8 bits with even, odd or no parity
- Up to two stop bits in 1/16 increments
- Error-detection capabilities

1.6.23 I^{2}C Module

- Interchip bus interface for EEPROMs, LCD controllers, A/D converters, and keypads
- Fully compatible with industry-standard I^{2}C bus
- Master or slave modes support multiple masters
- Automatic interrupt generation with programmable level

1.6.24 Interrupt Controllers

- Two interrupt controllers, supporting up to 64 interrupt sources each, organized as seven programmable levels
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source plus a global mask-all capability
- Support for service routine software interrupt acknowledge (IACK) cycles
- Combinational path to provide wake-up from low power modes

1.6.25 Edge Port Module

- Each pin can be individually configured as low level sensistive interrupt pin or edge-detecting interrupt pin (rising, falling, or both)
- Exit stop mode via level-detect function

1.6.26 DMA Controller

- 16 fully programmable channels with 32-byte transfer control
- Data movement via dual-address transfers for 8-, 16-, 32- and 128-bit data values
- Programmable source, destination addresses, transfer size, support for enhanced address modes
- Support for major and minor nested counters with one request and one interrupt per channel
- Support for channel-to-channel linking and scatter/gather for continuous transfers with fixed priority and round-robin channel arbitration
- External request pins for up to 2 channels

1.6.27 General Purpose I/O interface

- Up to 93 bits of GPIO for the MCF54450 and MCF54451
- Up to 132 bits of GPIO for the MCF54452, MCF54453, MCF54454, and MCF54455
- Bit manipulation supported via set/clear functions
- Various unused peripheral pins may be used as GPIO

1.6.28 System Debug Support

- Background debug mode (BDM) Revision D+
- Real time debug support, with four PC breakpoint registers and a pair of address breakpoint registers with optional data

1.6.29 JTAG Support

- JTAG part identification and part revision numbers

1.7 Memory Map Overview

Table 1-2 illustrates the overall memory map of the device.

Table 1-2. System Memory Map

Internal Address[31:28]	Address Range	Destination Slave	Slave Memory Size
00xx	0x0000_0000 – 0x3FFF_FFFF	FlexBus	1024 MB
01xx	0x4000_0000 – 0x7FFF_FFFF	SDRAM Controller	1024 MB
1000	0x8000_0000 – 0x8FFF_FFFF	Internal SRAM	256 MB
1001	0x9000_0000 – 0x9FFF_FFFF	ATA Controller	256 MB
101x	0xA000_0000 – 0xBFFF_FFFF	PCI Controller	512 MB
110x	0xC000_0000 – 0xDFFF_FFFF	FlexBus	512 MB
1110	0xE000_0000 – 0xEFFF_FFFF	Reserved	256 MB
1111	0xF000_0000 – 0xFFFF_FFFF	Internal Peripheral Space	256 MB

NOTE

This memory map provides two disjoint regions mapped to the FlexBus controller to support glueless connections to external memories (e.g., flash and SRAM), as well as a second space with one (or more) unique chip-selects that can be used for non-cacheable, non-memory devices (addresses 0xC000_0000 – 0xDFFF_FFFF). Additionally, this mapping is selected because it easily maps into the ColdFire access control registers, which provide a coarse association between memory addresses and their attributes (e.g., cacheable, non-cacheable). For this device, one possible configuration defines the default memory attribute as non-chacheable, and one ACR is then used to identify cacheable addresses. For example, ADDR[31] equaling 0 identifies the cacheable space.

1.7.1 Internal Peripheral Space

The internal peripheral space contains locations for all internal registers used to program and control the device's functional blocks and external interfaces. Table 1-3 summarizes the various register spaces and their base addresses. Each slot is 16 kB in size, which is not necessarily taken up entirely by the functional blocks. Any slot not illustrated is reserved. See corresponding chapter for details on their individual memory maps.

Table 1-3. Internal Peripheral Space Memory Map

Base Address	Slot Number	Peripheral
0xFC00_0000	0	SCM (MPR and PACRs)
0xFC00_4000	1	Crossbar switch
0xFC00_8000	2	FlexBus
0xFC03_0000	12	FEC0
0xFC03_4000	13	FEC1
0xFC03_C000	15	Real-Time Clock
0xFC04_0000	16	SCM (CWT and Core Fault Registers)
0xFC04_4000	17	eDMA Controller
0xFC04_8000	18	Interrupt Controller 0
0xFC04_C000	19	Interrupt Controller 1
0xFC05_4000	21	Interrupt Controller IACK
0xFC05_8000	22	I^{2}C
0xFC05_C000	23	DSPI
0xFC06_0000	24	UART0
0xFC06_4000	25	UART1
0xFC06_8000	26	UART2
0xFC07_0000	28	DMA Timer 0

Table 1-3. Internal Peripheral Space Memory Map (continued)

Base Address	Slot Number	Peripheral
0xFC07_4000	29	DMA Timer 1
0xFC07_8000	30	DMA Timer 2
0xFC07_C000	31	DMA Timer 3
0xFC08_0000	32	PIT 0
0xFC08_4000	33	PIT 1
0xFC08_8000	34	PIT 2
0xFC08_C000	35	PIT 3
0xFC09_4000	37	Edge Port
0xFC0A_0000	40	CCM, Reset Controller, Power Management
0xFC0A_4000	41	Pin Multiplexing and Control (GPIO)
0xFC0A_8000	42	PCI Controller
0xFC0A_C000	43	PCI Arbiter
0xFC0B_0000	44	USB On-the-Go
0xFC0B_4000	45	RNG
0xFC0B_8000	46	SDRAM Controller
0xFC0B_C000	47	SSI
0xFC0C_4000	49	PLL

1.8 Documentation

Documentation is available from a local Freescale distributor, a Freescale sales office, the Freescale Literature Distribution Center, or through the Freescale world-wide web address at http://www.freescale.com/coldfire.

Chapter 2
Signal Descriptions

2.1 Introduction

This chapter describes the external signals on the device. It includes an alphabetical signal listing of signals that characterizes each signal as an input or output, defines its state at reset, and identifies whether a pull-up resistor should be used.

NOTE

The terms assertion and negation are used to avoid confusion when dealing with a mixture of active-low and active-high signals. The term asserted indicates that a signal is active, independent of the voltage level. The term negated indicates that a signal is inactive.

Active-low signals, such as $\overline{\text{SD\_SRAS}}$ and $\overline{\text{TA}}$, are indicated with an overbar.

2.2 Signal Properties Summary

The below table lists the signals grouped by functionality.

NOTE

In this table and throughout this document, a single signal within a group is designated without square brackets (i.e., FB_AD23), while designations for multiple signals within a group use brackets (i.e., FB_AD[23:21]) and is meant to include all signals within the two bracketed numbers when these numbers are separated by a colon.

NOTE

The primary functionality of a pin is not necessarily its default functionality. Most pins that are muxed with GPIO default to their GPIO functionality. See Table 2-1 for a list of the exceptions.

Table 2-1. Special-Case Default Signal Functionality

Pin	256 MAPBGA	360 TEPBGA
FB_AD[31:0]	FB_AD[31:0] except when serial boot selects 0-bit boot port size.	
$\overline{\text{FB\_BE/BWE}}$[3:0]	$\overline{\text{FB\_BE/BWE}}$[3:0]	
$\overline{\text{FB\_CS}}$[3:1]	$\overline{\text{FB\_CS}}$[3:1]	

Table 2-1. Special-Case Default Signal Functionality (continued)

Pin	256 MAPBGA	360 TEPBGA
FB_OE	FB_OE	
FB_R/W	FB_R/W	
FB_TA	FB_TA	
FB_TS	FB_TS	
PCI_GNT[3:0]	GPIO	PCI_GNT[3:0]
PCI_REQ[3:0]	GPIO	PCI_REQ[3:0]
IRQ1	GPIO	PCI_INTA and configured as an agent.
ATA_RESET	GPIO	ATA reset

Table 2-2. MCF5445*x* Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
Reset								
RESET	—	—	—	U	I	EVDD	L4	Y18
RSTOUT	—	—	—	—	O	EVDD	M15	B17
Clock								
EXTAL/PCI_CLK	—	—	—	—	I	EVDD	M16	A16
XTAL	—	—	—	U[3]	O	EVDD	L16	A17
Mode Selection								
BOOTMOD[1:0]	—	—	—	—	I	EVDD	M5, M7	AB17, AB21
FlexBus								
FB_AD[31:24]	PFBADH[7:0][4]	FB_D[31:24]	—	—	I/O	EVDD	A14, A13, D12, C12, B12, A12, D11, C11	J2, K4, J1, K1–3, L1, L4
FB_AD[23:16]	PFBADMH[7:0][4]	FB_D[23:16]	—	—	I/O	EVDD	B11, A11, D10, C10, B10, A10, D9, C9	L2, L3, M1–4, N1–2
FB_AD[15:8]	PFBADML[7:0][4]	FB_D[15:8]	—	—	I/O	EVDD	B9, A9, D8, C8, B8, A8, D7, C7	P1–2, R1–3, P4, T1–2
FB_AD[7:0]	PFBADL[7:0][4]	FB_D[7:0]	—	—	I/O	EVDD	B7, A7, D6, C6, B6, A6, D5, C5	T3–4, U1–3, V1–2, W1
FB_BE/BWE[3:2]	PBE[3:2]	FB_TSIZ[1:0]	—	—	O	EVDD	B5, A5	Y1, W2
FB_BE/BWE[1:0]	PBE[1:0]	—	—	—	O	EVDD	B4, A4	W3, Y2

Table 2-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
FB_CLK	—	—	—	—	O	EVDD	B13	J3
FB_CS[3:1]	PCS[3:1]	—	—	—	O	EVDD	C2, D4, C3	W5, AA4, AB3
FB_CS0	—	—	—	—	O	EVDD	C4	Y4
FB_OE	PFBCTL3	—	—	—	O	EVDD	A2	AA1
FB_R/W	PFBCTL2	—	—	—	O	EVDD	B2	AA3
FB_TA	PFBCTL1	—	—	U	I	EVDD	B1	AB2
FB_TS	PFBCTL0	FB_ALE	FB_TBST	—	O	EVDD	A3	Y3
PCI Controller[5]								
PCI_AD[31:0]	—	FB_A[31:0]	—	—	I/O	EVDD	—	C11, D11, A10, B10, J4, G2, G3, F1, D12, C12, B12, A11, B11, B9, D9, D10, A8, B8, A5, B5, A4, A3, B3, D4, D3, E3–E1, F3, C2, D2, C1
—	—	FB_A[23:0]	—	—	I/O	EVDD	K14–13, J15–13, H13–15, G15–13, F14–13, E15–13, D16, B16, C15, B15, C14, D15, C16, D14	—
PCI_CBE[3:0]	—	—	—	—	I/O	EVDD	—	G4, E4, D1, B1
PCI_DEVSEL	—	—	—	—	O	EVDD	—	F2
PCI_FRAME	—	—	—	—	I/O	EVDD	—	B2
PCI_GNT3	PPCI7	ATA_DMACK	—	—	O	EVDD	—	B7
PCI_GNT[2:1]	PPCI[6:5]	—	—	—	O	EVDD	—	C8, C9
PCI_GNT0/ PCI_EXTREQ	PPCI4	—	—	—	O	EVDD	—	A9
PCI_IDSEL	—	—	—	—	I	EVDD	—	D5
PCI_IRDY	—	—	—	—	I/O	EVDD	—	C3
PCI_PAR	—	—	—	—	I/O	EVDD	—	C4
PCI_PERR	—	—	—	—	I/O	EVDD	—	B4
PCI_REQ3	PPCI3	ATA_INTRQ	—	—	I	EVDD	—	C7
PCI_REQ[2:1]	PPCI[2:1]	—	—	—	I	EVDD	—	D7, C5
PCI_REQ0/ PCI_EXTGNT	PPCI0	—	—	—	I	EVDD	—	A2

Table 2-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
PCI_RST	—	—	—	—	O	EVDD	—	B6
PCI_SERR	—	—	—	—	I/O	EVDD	—	A6
PCI_STOP	—	—	—	—	I/O	EVDD	—	A7
PCI_TRDY	—	—	—	—	I/O	EVDD	—	C10
SDRAM Controller								
SD_A[13:0]	—	—	—	—	O	SDVDD	R1, P1, N2, P2, R2, T2, M4, N3, P3, R3, T3, T4, R4, N4	V22, U20–22, T19–22, R20–22, N19, P20–21
SD_BA[1:0]	—	—	—	—	O	SDVDD	P4, T5	P22, P19
SD_CAS	—	—	—	—	O	SDVDD	T6	L19
SD_CKE	—	—	—	—	O	SDVDD	N5	N22
SD_CLK	—	—	—	—	O	SDVDD	T9	L22
SD_CLK	—	—	—	—	O	SDVDD	T8	M22
SD_CS[1:0]	—	—	—	—	O	SDVDD	P6, R6	L20, M20
SD_D[31:16]	—	—	—	—	I/O	SDVDD	N6, T7, N7, P7, R7, R8, P8, N8, N9, T10, R10, P10, N10, T11, R11, P11	L21, K22, K21, K20, J20, J19, J21, J22, H20, G22, G21, G20, G19, F22, F21, F20
SD_DM[3:2]	—	—	—	—	O	SDVDD	P9, N12	H21, E21
SD_DQS[3:2]	—	—	—	—	O	SDVDD	R9, N11	H22, E22
SD_RAS	—	—	—	—	O	SDVDD	P5	N21
SD_VREF	—	—	—	—	I	SDVDD	M8	M21
SD_WE	—	—	—	—	O	SDVDD	R5	N20
External Interrupts Port[6]								
IRQ7	PIRQ7	—	—	—	I	EVDD	L1	ABB13
IRQ4	PIRQ4	—	SSI_CLKIN	—	I	EVDD	L2	ABB13
IRQ3	PIRQ3	—	—	—	I	EVDD	L3	AB14
IRQ1	PIRQ1	PCI_INTA	—	—	I	EVDD	F15	C6
FEC0								
FEC0_MDC	PFECI2C3	—	—	—	O	EVDD	F3	AB8
FEC0_MDIO	PFECI2C2	—	—	—	I/O	EVDD	F2	Y7

Table 2-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
FEC0_COL	PFEC0H4	—	ULPI_DATA7	—	I	EVDD	E1	AB7
FEC0_CRS	PFEC0H0	—	ULPI_DATA6	—	I	EVDD	F1	AA7
FEC0_RXCLK	PFEC0H3	—	ULPI_DATA1	—	I	EVDD	G1	AA8
FEC0_RXDV	PFEC0H2	FEC0_RMII_CRS_DV	—	—	I	EVDD	G2	Y8
FEC0_RXD[3:2]	PFEC0L[3:2]	—	ULPI_DATA[5:4]	—	I	EVDD	G3, G4	AB9, Y9
FEC0_RXD1	PFEC0L1	FEC0_RMII_RXD1	—	—	I	EVDD	H1	W9
FEC0_RXD0	PFEC0H1	FEC0_RMII_RXD0	—	—	I	EVDD	H2	AB10
FEC0_RXER	PFEC0L0	FEC0_RMII_RXER	—	—	I	EVDD	H3	AA10
FEC0_TXCLK	PFEC0H7	FEC0_RMII_REF_CLK	—	—	I	EVDD	H4	Y10
FEC0_TXD[3:2]	PFEC0L[7:6]	—	ULPI_DATA[3:2]	—	O	EVDD	J1, J2	W10, AB11
FEC0_TXD1	PFEC0L5	FEC0_RMII_TXD1	—	—	O	EVDD	J3	AA11
FEC0_TXD0	PFEC0H5	FEC0_RMII_TXD0	—	—	O	EVDD	J4	Y11
FEC0_TXEN	PFEC0H6	FEC0_RMII_TXEN	—	—	O	EVDD	K1	W11
FEC0_TXER	PFEC0L4	—	ULPI_DATA0	—	O	EVDD	K2	AB12
FEC1								
FEC1_MDC	PFECI2C5	—	$\overline{\text{ATA\_DIOR}}$	—	O	EVDD	—	W20
FEC1_MDIO	PFECI2C4	—	$\overline{\text{ATA\_DIOW}}$	—	I/O	EVDD	—	Y22
FEC1_COL	PFEC1H4	—	ATA_DATA7	—	I	EVDD	—	AB18
FEC1_CRS	PFEC1H0	—	ATA_DATA6	—	I	EVDD	—	AA18
FEC1_RXCLK	PFEC1H3	—	ATA_DATA5	—	I	EVDD	—	W14
FEC1_RXDV	PFEC1H2	FEC1_RMII_CRS_DV	ATA_DATA15	—	I	EVDD	—	AB15
FEC1_RXD[3:2]	PFEC1L[3:2]	—	ATA_DATA[4:3]	—	I	EVDD	—	AA15, Y15
FEC1_RXD1	PFEC1L1	FEC1_RMII_RXD1	ATA_DATA14	—	I	EVDD	—	AA17
FEC1_RXD0	PFEC1H1	FEC1_RMII_RXD0	ATA_DATA13	—	I	EVDD	—	Y17
FEC1_RXER	PFEC1L0	FEC1_RMII_RXER	ATA_DATA12	—	I	EVDD	—	W17
FEC1_TXCLK	PFEC1H7	FEC1_RMII_REF_CLK	ATA_DATA11	—	I	EVDD	—	AB19
FEC1_TXD[3:2]	PFEC1L[7:6]	—	ATA_DATA[2:1]	—	O	EVDD	—	Y19, W18

Table 2-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
FEC1_TXD1	PFEC1L5	FEC1_RMII_TXD1	ATA_DATA10	—	O	EVDD	—	AA19
FEC1_TXD0	PFEC1H5	FEC1_RMII_TXD0	ATA_DATA9	—	O	EVDD	—	Y20
FEC1_TXEN	PFEC1H6	FEC1_RMII_TXEN	ATA_DATA8	—	O	EVDD	—	AA21
FEC1_TXER	PFEC1L4	—	ATA_DATA0	—	O	EVDD	—	AA22
USB On-the-Go								
USB_DM	—	—	—	—	O	USB VDD	F16	A14
USB_DP	—	—	—	—	O	USB VDD	E16	A15
USB_VBUS_EN	PUSB1	USB_PULLUP	ULPI_NXT	—	O	USB VDD	E5	AA2
USB_VBUS_OC	PUSB0	—	ULPI_STP	UD[7]	I	USB VDD	B3	V4
ATA								
ATA_BUFFER_EN	PATAH5	—	—	—	O	EVDD	—	Y13
ATA_CS[1:0]	PATAH[4:3]	—	—	—	O	EVDD	—	W21, W22
ATA_DA[2:0]	PATAH[2:0]	—	—	—	O	EVDD	—	V19–21
ATA_RESET	PATAL2	—	—	—	O	EVDD	—	W13
ATA_DMARQ	PATAL1	—	—	—	I	EVDD	—	AA14
ATA_IORDY	PATAL0	—	—	—	I	EVDD	—	Y14
Real Time Clock								
EXTAL32K	—	—	—	—	I	EVDD	J16	A13
XTAL32K	—	—	—	—	O	EVDD	H16	A12
SSI								
SSI_MCLK	PSSI4	—	—	—	O	EVDD	T13	D20
SSI_BCLK	PSSI3	U1CTS	—	—	I/O	EVDD	R13	E19
SSI_FS	PSSI2	U1RTS	—	—	I/O	EVDD	P12	E20
SSI_RXD	PSSI1	U1RXD	—	UD	I	EVDD	T12	D21
SSI_TXD	PSSI0	U1TXD	—	UD	O	EVDD	R12	D22
I^{2}C								
I2C_SCL	PFECI2C1	—	U2TXD	U	I/O	EVDD	K3	AA12

Table 2-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
I2C_SDA	PFECI2C0	—	U2RXD	U	I/O	EVDD	K4	Y12
DMA								
DACK1	PDMA3	—	ULPI_DIR	—	O	EVDD	M14	C17
DREQ1	PDMA2	—	USB_CLKIN	U	I	EVDD	P16	C18
DACK0	PDMA1	DSPI_PCS3	—	—	O	EVDD	N15	A18
DREQ0	PDMA0	—	—	U	I	EVDD	N16	B18
DSPI								
DSPI_PCS5/PCSS	PDSPI6	—	—	—	O	EVDD	N14	D18
DSPI_PCS2	PDSPI5	—	—	—	O	EVDD	L13	A19
DSPI_PCS1	PDSPI4	SBF_CS	—	—	O	EVDD	P14	B20
DSPI_PCS0/SS	PDSPI3	—	—	U	I/O	EVDD	R16	D17
DSPI_SCK	PDSPI2	SBF_CK	—	—	I/O	EVDD	R15	A20
DSPI_SIN	PDSPI1	SBF_DI	—	[8]	I	EVDD	P15	B19
DSPI_SOUT	PDSPI0	SBF_DO	—	—	O	EVDD	N13	C20
UARTs								
U1CTS	PUART7	—	—	—	I	EVDD	—	V3
U1RTS	PUART6	—	—	—	O	EVDD	—	U4
U1RXD	PUART5	—	—	—	I	EVDD	—	P3
U1TXD	PUART4	—	—	—	O	EVDD	—	N3
U0CTS	PUART3	—	—	—	I	EVDD	M3	Y16
U0RTS	PUART2	—	—	—	O	EVDD	M2	AA16
U0RXD	PUART1	—	—	—	I	EVDD	N1	AB16
U0TXD	PUART0	—	—	—	O	EVDD	M1	W15
Note: The UART1 and UART 2 signals are multiplexed on the DMA timers and I2C pins.								
DMA Timers								
DT3IN	PTIMER3	DT3OUT	U2RXD	—	I	EVDD	C13	H2
DT2IN	PTIMER2	DT2OUT	U2TXD	—	I	EVDD	D13	H1
DT1IN	PTIMER1	DT1OUT	U2CTS	—	I	EVDD	B14	H3
DT0IN	PTIMER0	DT0OUT	U2RTS	—	I	EVDD	A15	G1

Table 2-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
BDM/JTAG[9]								
PSTDDATA[7:0]	—	—	—	—	O	EVDD	E2, D1, F4, E3, D2, C1, E4, D3	AA6, AB6, AB5, W6, Y6, AA5, AB4, Y5
JTAG_EN	—	—	—	D	I	EVDD	M11	C21
PSTCLK	—	TCLK	—	—	I	EVDD	P13	C22
DSI	—	TDI	—	U	I	EVDD	T15	C19
DSO	—	TDO	—	—	O	EVDD	T14	A21
$\overline{\text{BKPT}}$	—	TMS	—	U	I	EVDD	R14	B21
DSCLK	—	$\overline{\text{TRST}}$	—	U	I	EVDD	M13	B22
Test								
TEST	—	—	—	D	I	EVDD	M6	AB20
PLLTEST	—	—	—	—	O	EVDD	K16	D15
Power Supplies								
IVDD	—	—	—	—	—	—	E6–12, F5, F12	D6, D8, D14, F4, H4, N4, R4, W4, W7, W8, W12, W16, W19
EVDD	—	—	—	—	—	—	G5, G12, H5, H12, J5, J12, K5, K12, L5–6, L12	D13, D19, G8, G11, G14, G16, J7, J16, L7, L16, N16, P7, R16, T8, T12, T14, T16
SD_VDD	—	—	—	—	—	—	L7–11, M9, M10	F19, H19, K19, M19, R19, U19
VDD_OSC	—	—	—	—	—	—	L14	B16
VDD_A_PLL	—	—	—	—	—	—	K15	C14
VDD_RTC	—	—	—	—	—	—	M12	C13
VSS	—	—	—	—	—	—	A1, A16, F6–11, G6–11, H6–11, J6–11, K6–11, T1, T16	A1, A22, B14, G7, G9–10, G12–13, G15, H7, H16, J9–14, K7, K9–14, K16, L9–14, M7, M9–M14, M16, N9–14, P9–14, P16, R7, T7, T9–11, T13, T15, AB1, AB22
VSS_OSC	—	—	—	—	—	—	L15	C16

[1] Pull-ups are generally only enabled on pins with their primary function, except as noted.

[2] Refers to pin's primary function.

[3] Enabled only in oscillator bypass mode (internal crystal oscillator is disabled).

[4] Serial boot must select 0-bit boot port size to enable the GPIO mode on these pins.

[5] When the PCI is enabled, all PCI bus pins come up configured as such. This includes the PCI_GNT and PCI_REQ lines, which have GPIO. The IRQ1/$\overline{\text{PCI\_INTA}}$ signal is a special case. It comes up as $\overline{\text{PCI\_INTA}}$ when booting as a PCI agent and as GPIO when booting as a PCI host.
For the 360 TEPBGA, booting with PCI disabled results in all dedicated PCI pins being safe-stated. The $\overline{\text{PCI\_GNT}}$ and $\overline{\text{PCI\_REQ}}$ lines and IRQ1/$\overline{\text{PCI\_INTA}}$ come up as GPIO.

[6] GPIO functionality is determined by the edge port module. The pin multiplexing and control module is only responsible for assigning the alternate functions.

[7] Depends on programmed polarity of the USB_VBUS_OC signal.

[8] Pull-up when the serial boot facility (SBF) controls the pin

[9] If JTAG_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The pin multiplexing and control module is not responsible for assigning these pins.

NOTE

2.3 Signal Primary Functions

2.3.1 Reset Signals

Table 2-3 describes signals used to reset the chip or to indicate a reset.

Table 2-3. Reset Signals

Signal Name	Abbreviation	Function	I/O
Reset In	$\overline{\text{RESET}}$	Primary reset input to the device. Asserting $\overline{\text{RESET}}$ resets the core and peripherals after four FB_CLK cycles. Asserting $\overline{\text{RESET}}$ also causes $\overline{\text{RSTOUT}}$ to be asserted.	I
Reset Out	$\overline{\text{RSTOUT}}$	Reset output ($\overline{\text{RSTOUT}}$) is an indicator that the chip is in reset. $\overline{\text{RSTOUT}}$ is asserted at least 512 internal system bus clock cycles (256 FB_CLK cycles) in response to any internal or external reset. (The exact time depends on how long it takes for the PLL to lock and/or the serial boot sequence to complete.)	O

2.3.2 PLL and Clock Signals

Table 2-4 describes signals that are used to support the on-chip clock generation circuitry.

Table 2-4. PLL and Clock Signals

Signal Name	Abbreviation	Function	I/O
External Clock In	EXTAL	Always driven by an external clock input except when used as a connection to the external crystal if the internal oscillator circuit is used. Clock source may be configured during reset. See Chapter 11, "Chip Configuration Module (CCM)," for more details. **Note:** This signal is also PCI_CLK (33 or 66 MHz) when running from an external oscillator with PCI enabled.	I
Crystal	XTAL	Used as a connection to the external crystal when the internal oscillator circuit is used to drive the crystal.	O
RTC External Clock In	EXTAL32K	Crystal input clock for the real-time clock module.	I
RTC Crystal	XTAL32K	Oscillator output to EXTAL RTC crystal.	O
FlexBus Clock Out	FB_CLK	Reflects one-half of the internal bus clock (or one-fourth the core/system clock). ($f_{sys/4}$)	O
USB Clock In	USB_CLKIN	This pin allows the user to drive the reference clock to the USB module as an alternate method of generating the USB reference clock during FS/LS operation. This pin should be driven only with a 60 MHz clock. When using the ULPI USB interface, this pin is the ULPI input clock.	I
SSI Clock In	SSI_CLKIN	This pin allows the user to drive a specific clock frequency to the SSI module.	I

2.3.3 Mode Selection

Table 2-5. Mode Selection Signals

Signal Name	Abbreviation	Function	I/O
Boot mode	BOOTMOD[1:0]	Indicates the device's boot mode and chip configuration at reset. See Chapter 11, "Chip Configuration Module (CCM)," for the signal encodings.	I

2.3.4 FlexBus Signals

Table 2-6 describes signals that are used for performing transactions on the external bus.

Table 2-6. FlexBus Signals

Signal Name	Abbreviation	Function	I/O
Address/Data Bus	FB_AD[31:0]	Defines address and data of external byte, word, and longword accesses. This three-state, bi-directional bus is the general-purpose address/data path to external SRAM and flash devices.	I/O
Byte Enables	FB_BE/BWE[3:0]	Defines flow of data on data bus. During peripheral accesses, these output signals indicate that data is to be latched or driven onto a byte of the data bus when driven low. The BE/BWE[3:0] signals are asserted only to the memory bytes used during a read or write access. BE/BWE0 controls access to the most significant byte lane of data, and BE/BWE3 controls access to the least significant byte lane of data. For SRAM or Flash devices, the BE/BWE*n* outputs should be connected to individual byte strobe signals. The BE/BWE*n* signals are asserted during accesses to on-chip peripherals, but not to on-chip SRAM or cache.	O
Output Enable	FB_OE	Indicates when an external device can drive data during external read cycles.	O
Transfer Acknowledge	FB_TA	Indicates external data transfer is complete. During a read cycle, when the processor recognizes TA, it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes TA, the bus cycle is terminated.	I
Read/Write	FB_R/W	Indicates direction of the data transfer on the bus for SRAM (R/W) accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device.	O
Transfer Size	FB_TSIZ[1:0]	Indicates bus width (8, 16, or 32 bits) for each chip select. The initial width for the bootstrap program chip select is determined by the initial state of TSIZ[1:0].	O
Transfer Burst	FB_TBST	Indicates external bus access is a burst access.	O
Transfer Start	FB_TS	Bus control output signal indicating the start of a transfer.	O
Address Latch Enable	FB_ALE	Indicates device has begun a bus transaction and the address and attributes are valid. FB_ALE is asserted for one bus clock cycle. In multiplexed mode, ALE is used externally as an address latch enable to capture the address phase of the bus transfer.	O
Chip Selects	FB_CS[3:0]	Select external devices for external bus transactions.	O

2.3.5 SDRAM Controller Signals

Table 2-7 describes signals used for SDRAM accesses.

Table 2-7. SDRAM Controller Signals

Signal Name	Abbreviation	Function	I/O
SDRAM Address Bus	SD_A[13:0]	Address bus used for multiplexed row and column addresses during SDRAM bus cycles.	O
SDRAM Data Bus	SD_D[31:16]	Bidirectional, non-multiplexed data bus for SDRAM accesses.	I/O
SDRAM Bank Address	SD_BA[1:0]	Selects one of the four SDRAM row banks.	O
SDRAM Clock Enable	SD_CKE	SDRAM clock enable.	O
DDR SDRAM Clock	SD_CLK	Output clock for DDR SDRAM.	O
DDR SDRAM Clock	$\overline{\text{SD\_CLK}}$	Inverted output clock for DDR SDRAM.	O
SDRAM Chip Selects	$\overline{\text{SD\_CS}}$[1:0]	SDRAM chip select signals.	O
DDR SDRAM Data Strobes	SD_DQS[3:2]	Indicates when valid data is on data bus.	I/O
SDRAM Write Data Byte Mask	SD_DQM[3:2]	Used to determine which byte lanes of data bus should be latched during a write cycle. The SD_DQM*n* should be connected to individual SDRAM DQM signals. Most SDRAMs associate DQM3 with the MSB, in which case SD_DQM3 should be connected to the SDRAM's DQM3 input.	O
SDRAM Column Address Strobe	$\overline{\text{SD\_CAS}}$	SDRAM column address strobe.	O
SDRAM Row Address Strobe	$\overline{\text{SD\_RAS}}$	SDRAM row address strobe.	O
SDRAM Write Enable	$\overline{\text{SD\_WE}}$	Indicates direction of data transfer on bus for SDRAM accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device.	O
SDRAM Voltage Reference	SD_VREF	Reference voltage for differential I/O pad cells. Should be half the voltage of the memory used in the system. For example, 2.5 V DDR results in an SD_VREF of 1.25 V. See the device's datasheet for the voltages and tolerances for the various memory modes.	I

2.3.6 PCI Controller Signals

Table 2-8 describes the external interrupt signals used on the external PCI bus.

Table 2-8. PCI Controller Signals

Signal Name	Abbreviation	Function	I/O
PCI Address/Data Bus	PCI_AD[31:0]	Multiplexed address/data bus.	I/O
PCI Command/Byte Enables	PCI_CBE[3:0]	Multiplexed PCI command and byte enables. The PCI command is present during address phase; the byte enables are present during data phase.	I/O
PCI Device Select	PCI_DEVSEL	Indicates processor has recognized itself as the target of a PCI transaction from address presented on the PCI bus.	O
PCI Frame	PCI_FRAME	Asserted by a PCI initiator to indicate the beginning of a transaction. It is negated when initiator is ready to complete final data phase.	I/O
PCI External Bus Grant	PCI_GNT[3:1]	Asserted to an external master to give it control of PCI bus. If internal PCI arbiter is enabled, it asserts one of the PCI_GNT[3:1] signals to grant ownership of PCI bus to external master. When PCI arbiter is disabled, PCI_GNT[3:1] are driven high and should be ignored.	O
PCI External Bus Grant/Request	PCI_GNT0/ PCI_EXTREQ	Asserted to external master device 0 to give it control of the PCI bus. When the PCI arbiter is disabled, the signal operates as the PCI_EXTREQ output, which is asserted when the processor needs to initiate a PCI transaction.	O
PCI Initialization Device Select	PCI_IDSEL	Asserted during a PCI type-0 configuration cycle to address the PCI configuration header.	O
PCI Initiator Ready	PCI_IRDY	Indicates that PCI initiator is ready to transfer data. During a write operation, assertion indicates the master is driving valid data on bus. During a read operation assertion indicates that master is ready to accept data.	I/O
PCI Parity	PCI_PAR	Indicates the parity of the data on the PCI_AD[31:0] and PCI_CBE[3:0] signals.	I/O
PCI Parity Error	PCI_PERR	Asserted when data phase parity error is detected if enabled.	I/O
PCI External Bus Request	PCI_REQ[3:1]	Asserted by an external PCI master when it requires access to the PCI bus.	I
PCI External Bus Request/Grant	PCI_REQ0/ PCI_EXTGNT	Asserted by external PCI master device 0 when it requires access to the PCI bus. When internal PCI arbiter is disabled, this signal is used as a grant input for PCI bus, which is driven by an external PCI arbiter.	I
PCI Reset	PCI_RST	Asserted by processor to reset PCI bus. It is asserted when processor is reset and must be negated to enable usage on PCI bus.	O
PCI System Error	PCI_SERR	Indicates detection of an address-phase-parity error.	I/O
PCI Stop	PCI_STOP	Indicates that the currently addressed target wishes to stop the current transaction.	I/O
PCI Target Ready	PCI_TRDY	Indicates currently addressed target is ready to complete the current data phase.	I/O
PCI Interrupt A	PCI_INTA	This output is the PCI interrupt A signal.	O

2.3.7 Serial Boot Facility Signals

Table 2-9. SBF Signals

Signal Name	Abbreviation	Function	I/O
SBF Chip Select	$\overline{\text{SBF\_CS}}$	Chip select used to access external SPI memory.	O
SBF Clock	SBF_CK	25 MHz clock source for external SPI memory.	O
SBF Data In	SBF_DI	Data being driven by SPI memory.	I
SBF Data Out	SBF_DO	Data out to SPI memory. SBF uses this output solely for the purpose of issuing the SPI memory READ command. SBF does not write data to SPI memory.	O

2.3.8 External Interrupt Signals

Table 2-10. External Interrupt Signals

Signal Name	Abbreviation	Function	I/O
External Interrupts	$\overline{\text{IRQ}}$[7,4,3,1]	External interrupt sources.	I

2.3.9 DMA Signals

Table 2-11. DMA Signals

Signal Name	Abbreviation	Function	I/O
DMA Request	$\overline{\text{DREQ}}$[1:0]	Asserted by an external device to request a DMA transfer.	I
DMA Acknowledge	$\overline{\text{DACK}}$[1:0]	Asserted by processor to indicate DMA request has been recognized.	O

2.3.10 Fast Ethernet Controller (FEC0 and FEC1) Signals

The following signals are used by the two Ethernet modules.

Table 2-12. Ethernet Module (FEC) Signals

Signal Name	Abbreviation	Function	I/O
Management Data	FEC*n*_MDIO	Transfers control information between external PHY and the media-access controller. Data is synchronous to FEC*n*_MDC. Applies to MII mode operation. This signal is an input after reset. When the FEC is operated in 10Mbps 7-wire interface mode, this signal should be connected to VSS.	I/O
Management Data Clock	FEC*n*_MDC	In Ethernet mode, FEC*n*_MDC is an output clock that provides a timing reference to PHY for data transfers on FEC*n*_MDIO signal. Applies to MII mode operation.	O
Collision	FEC*n*_COL	Asserted upon collision detection and remains asserted while collision persists. This signal is not defined for full-duplex mode.	I
Carrier Receive Sense	FEC*n*_CRS	When asserted, indicates transmit or receive medium is not idle. Applies to MII mode operation.	I

Table 2-12. Ethernet Module (FEC) Signals (continued)

Signal Name	Abbreviation	Function	I/O
Transmit Clock	FEC*n*_TXCLK	Input clock providing a timing reference for FEC*n*_TXEN, FEC*n*_TXD[3:0] and FEC*n*_TXER	I
Transmit Enable	FEC*n*_TXEN	Indicates when valid nibbles are present on MII. This signal is asserted with the first nibble of a preamble and is negated before the first FEC*n*_TXCLK following the final nibble of the frame.	O
Transmit Data 0	FEC*n*_TXD0	FEC*n*_TXD0 is the serial output Ethernet data and is valid only during the assertion of FEC*n*_TXEN. This signal is used for 10-Mbps Ethernet data. Also used for MII mode data in conjunction with FEC*n*_TXD[3:1].	O
Transmit Data 1–3	FEC*n*_TXD[3:1]	In Ethernet mode, these pins contain serial output Ethernet data and are valid only during assertion of FEC*n*_TXEN in MII mode.	O
Transmit Error	FEC*n*_TXER	In Ethernet mode, when FEC*n*_TXER is asserted for one or more clock cycles while FEC*n*_TXEN is also asserted, the PHY sends one or more illegal symbols. FEC*n*_TXER has no effect at 10 Mbps or when FEC*n*_TXEN is negated. Applies to MII mode operation.	O
Receive Clock	FEC*n*_RXCLK	Provides a timing reference for FEC*n*_RXDV, FEC*n*_RXD[3:0], and FEC*n*_RXER.	I
Receive Data Valid	FEC*n*_RXDV	Asserting the FEC*n*_RXDV input indicates that the PHY has valid nibbles present on the MII. FEC*n*_RXDV should remain asserted from the first recovered nibble of the frame through to the last. Assertion of FEC*n*_RXDV must start no later than the SFD and exclude any EOF.	I
Receive Data 0	FEC*n*_RXD0	FEC*n*_RXD0 is the Ethernet input data transferred from the PHY to the media-access controller when FEC*n*_RXDV is asserted. This signal is used for 10-Mbps Ethernet data. This signal is also used for MII mode Ethernet data in conjunction with FEC*n*_RXD[3:1].	I
Receive Data 1–3	FEC*n*_RXD[3:1]	In Ethernet mode, these pins contain Ethernet input data transferred from the PHY to the media access controller when FEC*n*_RXDV is asserted in MII mode operation.	I
Receive Error	FEC*n*_RXER	In Ethernet mode, when asserted with FEC*n*_RXDV, FEC*n*_RXER indicates that the PHY has detected an error in current frame. When FEC*n*_RXDV is not asserted FEC*n*_RXER has no effect. Applies to MII mode operation.	I

2.3.11 I^{2}C I/O Signals

Table 2-13. I^{2}C I/O Signals

Signal Name	Abbreviation	Function	I/O
Serial Clock	I2C_SCL	Open-drain clock signal for I^{2}C interface. It is driven by the I^{2}C module when the bus is in master mode, or it becomes the clock input when the I^{2}C is in slave mode.	I/O
Serial Data	I2C_SDA	Open-drain signal serving as the data input/output for the I^{2}C interface.	I/O

2.3.12 ATA Controller Signals

Table 2-14. ATA Controller Signals

Signal Name	Abbreviation	Function	I/O
ATA Data Bus	ATA_DATA[15:0]	The bi-directional, three-state ATA data bus.	I/O
ATA Buffer Enable	ATA_BUFFER_EN	This output signal is the ATA transceiver direction-control signal.	O
ATA Chip Selects	$\overline{\text{ATA\_CS}}$[1:0]	These output signals ATA bus chip selects.	O
ATA Address	ATA_DA[2:0]	These output signals are ATA bus address group.	O
ATA Reset	$\overline{\text{ATA\_RESET}}$	This output signal is ATA reset signal. When asserted, ATA bus is in reset state. When negated, no reset. ATA bus is in reset when the appropriate bit in the control register is cleared. After system reset, ATA bus is in reset.	O
ATA DMA Request	ATA_DMARQ	This input signal is the ATA bus device DMA request. It is asserted by the device if it wants to transfer data using multiword DMA or ultra DMA mode	I
ATA DMA Acknowledge	$\overline{\text{ATA\_DMACK}}$	This output signal is the ATA bus host DMA acknowledge. It is asserted by the host when it grants the DMA request.	O
ATA I/O Ready In	ATA_IORDY	This input is the ATA IORDY line. It has three functions: • IORDY—active low wait during PIO cycles, • DDMARDY—active low device ready during ultra DMA out transfers • DSTROBE—device strobe during ultra DMA in transfers	I
ATA DIO Read	$\overline{\text{ATA\_DIOR}}$	This output signal corresponds to ATA signal DIOR. During PIO and multiword DMA transfers, its function is read strobe. During ultra DMA IN burst, its function is HDMARDY. During ultra DMA OUT burst, its function is host strobe (HSTROBE).	O
ATA DIO Write	$\overline{\text{ATA\_DIOW}}$	This output signal corresponds to ATA signal DIOW. During PIO and multiword DMA transfers, its function is write strobe. During ultra DMA burst, its function is STOP, signalling when the host wants to terminate an ultra DMA transfer.	O
ATA Interrupt Request	ATA_INTRQ	This input signal is the ATA bus interrupt request. It is asserted by the device when it wants to interrupt.	I

2.3.13 DMA Serial Peripheral Interface (DSPI) Signals

Table 2-15. DMA Serial Peripheral Interface (DSPI) Signals

Signal Name	Abbreviation	Function	I/O
DSPI Synchronous Serial Output	DSPI_SOUT	Provides the serial data from the DSPI and can be programmed to be driven on the rising or falling edge of DSPI_SCK. Each byte is sent msb first.	O
DSPI Synchronous Serial Data Input	DSPI_SIN	Provides the serial data to the DSPI and can be programmed to be sampled on the rising or falling edge of DSPI_SCK. Each byte is written to RAM lsb first.	I
DSPI Serial Clock	DSPI_SCK	Provides the serial clock from the DSPI. In master mode, the processor generates DSPI_SCK, while in slave mode, DSPI_SCK is an input from an external bus master.	I/O
DSPI Peripheral Chip Select 5/Peripheral Chip Select Strobe	DSPI_PCS5/ DSPI_PCSS	When in master mode and the DSPI_MCR[PCSSE] bit cleared, DSPI_PCS5 is a peripheral chip select output that selects which slave device the current transmission is intended. DSPI_PCSS provides a strobe signal that can be used with an external demultiplexer for deglitching of the DSPI_PCS*n* signals. When in master mode and the DSPI_MCR[PCSSE] bit is set, DSPI_PCSS provides the appropriate timing for the decoding of the DSPI_PCS[3:0] signals, which prevents glitches from occurring. In slave mode, this signal is not used.	O
DSPI Peripheral Chip Selects	DSPI_PCS[3:1]	Provide DSPI peripheral chip selects that can be programmed to be active high or low.	O
DSPI Peripheral Chip Select 0/Slave Select	DSPI_PCS0/ DSPI_SS	In master mode, DSPI_PCS0 is a peripheral chip select output that selects which slave device the current transmission is intended. In slave mode, the SS signal is a slave select input that allows an SPI master to select the processor as the target for transmission.	I/O

2.3.14 Synchronous Serial Interface (SSI) Signals

Table 2-16. SSI Module Signals

Signal Name	Abbreviation	Function	I/O
Serial Bit Clock	SSI_BCLK	Used by the receive and transmit blocks. In gated clock mode, SSI_BCLK is only valid during transmission of data, otherwise it is pulled to an inactive state.	I/O
Serial Master Clock	SSI_MCLK	This clock signal is output from the device when it is the master. When in I^{2}S master mode, this signal is referred to as the oversampling clock. The frequency of SSI_MCLK is a multiple of the frame clock.	O
Serial Frame Sync	SSI_FS	Used by transmitter/receiver to synchronize the transfer of data. In gated clock mode, this signal is not used. When configured as an input, the external device should drive SSI_FS during the rising edge of SSI_BCLK.	I/O
Serial Receive Data	SSI_RXD	Receives data into the receive data shift register	I
Serial Transmit Data	SSI_TXD	Transmits data from the serial transmit shift register.	O

2.3.15 Universal Serial Bus (USB) Signals

Table 2-17. USB Module Signals

Signal Name	Abbreviation	Function	I/O
USB D-	USB_DM	D- output of the dual-speed transceiver for the On-the-Go module.	O
USB D+	USB_DP	D+ output of the dual-speed transceiver for the On-the-Go module.	O
USB VBUS Enable	USB_VBUS_EN	Enables the off-chip VBUS charge pump when USB OTG module is configured as a host.	O
USB VBUS over-current	USB_VBUS_OC	Indicates to the processor that a short has occurred on USB data bus.	I
USB External Pull-up Enable	USB_PULLUP	Either use this pullup enable output signal, or turn it off in the CCM's MISCCR[USBPUE] bit. If internal pullup (and not this output signal) is used, the internal pullup automatically switches impedances based on whether USB is transmitting or receiving.	O
ULPI Data Bus	ULPI_DATA[7:0]	These bi-directional signals are ULPI data bus. Synchronous to USB_CLKIN.	I/O
ULPI Next Data	ULPI_NXT	This input is the ULPI next data. Synchronous to USB_CLKIN.	I
ULPI Stop Data	ULPI_STP	This output is the ULPI stop data. Synchronous to USB_CLKIN.	O
ULPI Data Bus Direction	ULPI_DIR	This input is the ULPI data bus direction. Synchronous to USB_CLKIN.	I

2.3.16 UART Module Signals

Table 2-18 describes the signals of the three UART modules, where *n* equals 0 – 2. Baud-rate clock inputs are not supported.

Table 2-18. UART Module Signals

Signal Name	Abbreviation	Function	I/O
Transmit Serial Data Output	U*n*TXD	Data is shifted out lsb first at the falling edge of the serial clock source. Output is held high when transmitter is disabled, idle, or in local loopback mode.	O
Receive Serial Data Input	U*n*RXD	Data is sampled lsb first at the serial clock source's rising edge. When the UART clock is stopped for power-down mode, any transition on this pin restarts it.	I
Clear-to-Send	$\overline{\mathrm{U}\mathit{n}\mathrm{CTS}}$	Indicates UART modules can begin data transmission	I
Request-to-Send	$\overline{\mathrm{U}\mathit{n}\mathrm{RTS}}$	Automatic request-to-send outputs from UART modules. They may also be asserted and negated as a function of the received FIFO level.	O

2.3.17 DMA Timer Signals

Table 2-19 describes the signals of the four DMA timer modules, where *n* equals 0 – 3.

Table 2-19. DMA Timer Signals

Signal Name	Abbreviation	Function	I/O
DMA Timer *n* Input	DT*n*IN	Can be programmed to cause events in the respective timer. It can clock the event counter or provide a trigger to the timer value capture logic.	I
DMA Timer *n* Output	DT*n*OUT	Output from respective timer.	O

2.3.18 Debug Support Signals

These signals are used as the interface to the on-chip JTAG controller and the BDM logic. Pin functionality between JTAG and BDM is dependent upon the JTAG_EN pin.

Table 2-20. Debug Support Signals

Signal Name	Abbreviation	Function	I/O
JTAG Enable	JTAG_EN	Enables JTAG (asserted) or BDM (negated) operation.	I
JTAG Signals			
Test Reset	$\overline{\text{TRST}}$	Active-low signal used to initialize the JTAG logic asynchronously.	I
Test Clock	TCLK	Used to synchronize the JTAG logic.	I
Test Mode Select	TMS	Used to sequence the JTAG state machine. TMS is sampled on the rising edge of TCLK.	I
Test Data Input	TDI	Serial input for test instructions and data. TDI is sampled on the rising edge of TCLK.	I
Test Data Output	TDO	Serial output for test instructions and data. TDO is three-stateable and actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O
BDM Signals			
Development Serial Clock	DSCLK	Clocks the serial communication port to the BDM module during packet transfers.	I
Breakpoint	$\overline{\text{BKPT}}$	Used to request a manual breakpoint.	I
Development Serial Input	DSI	Internally-synchronized signal provides data input for the serial communication port to the BDM module.	I
Development Serial Output	DSO	Internally-registered signal provides serial output communication for BDM module responses.	O
Processor Status Clock	PSTCLK	Used by the development system to know when to sample DDATA and PST signals.	O
Processor Status/ Debug Data	PSTDDATA[7:0]	Display captured processor status and captured address/data values. These outputs change on the negative edge of PSTCLK.	O

Table 2-21. Processor Status

PST[3:0]	Processor Status
0000	Continue execution
0001	Begin execution of one instruction
0010	Reserved
0011	Entry into user mode
0100	Begin execution of PULSE and WDDATA instructions
0101	Begin execution of taken branch
0110	Reserved
0111	Begin execution of RTE instruction
1000	Begin one-byte transfer on PSTDDATA
1001	Begin two-byte transfer on PSTDDATA
1010	Begin three-byte transfer on PSTDDATA
1011	Begin four-byte transfer on PSTDDATA
1100	Exception processing
1101	Reserved
1110	Processor is stopped
1111	Processor is halted

2.3.19 Test Signals

Table 2-22 describes test signals reserved for factory testing.

Table 2-22. Test Signals

Signal Name	Abbreviation	Function	I/O
Test	TEST	Reserved for factory testing only and in normal modes of operation should be connected to VSS to prevent unintentional activation of test functions.	I
PLL Test	PLL_TEST	Reserved for factory testing only and should be treated as a no-connect (NC).	O

2.3.20 Power and Ground Pins

The pins described in Table 2-23 provide system power and ground to the device. Multiple pins are provided for adequate current capability. All power supply pins must have adequate bypass capacitance for high-frequency noise suppression.

Table 2-23. Power and Ground Pins

Signal Name	Abbreviation	Function	I/O
PLL Analog Supply	VDD_A_PLL	Dedicated power supply signal to isolate the sensitive PLL analog (VCO) circuitry from the normal levels of noise present on the digital power supply.	—
Oscillator	VDD_OSC VSS_OSC	Dedicated power supply signals to isolate the sensitive oscillator circuitry from the normal levels of noise present on the digital power supply.	—
Positive I/O Supply	EVDD	These pins supply positive power to the I/O pads.	—
Positive Core Supply	IVDD	These pins supply positive power to the core logic.	—
SDRAMC Supply	SD_VDD	These pins supply positive power to the SDRAM controller.	—
USB Supply	USB_VDD	These pins supply positive power to the USB controller.	—
Real-time clock Supply	RTC_VDD	These pins supply positive power to the RTC module.	—
Ground	VSS	These pins are the negative supply (ground) for the device.	—

2.4 External Boot Mode

After reset the address bus, data bus, FlexBus control signals, and SDRAM control signals default to their bus functionalities. All other signals default to GPIO inputs (if applicable).

Chapter 3
ColdFire Core

3.1 Introduction

This section describes the organization of the Version 4 (V4) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, see the ISA_C definition in the *ColdFire Family Programmer's Reference Manual*. The V4 ColdFire core includes the enhanced multiply-accumulate unit (EMAC), and memory management unit (MMU), which are explained in detail in their own chapters. This chapter also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings.

3.1.1 Overview

As with all ColdFire cores, the V4 ColdFire core is comprised of two separate pipelines decoupled by an instruction buffer.

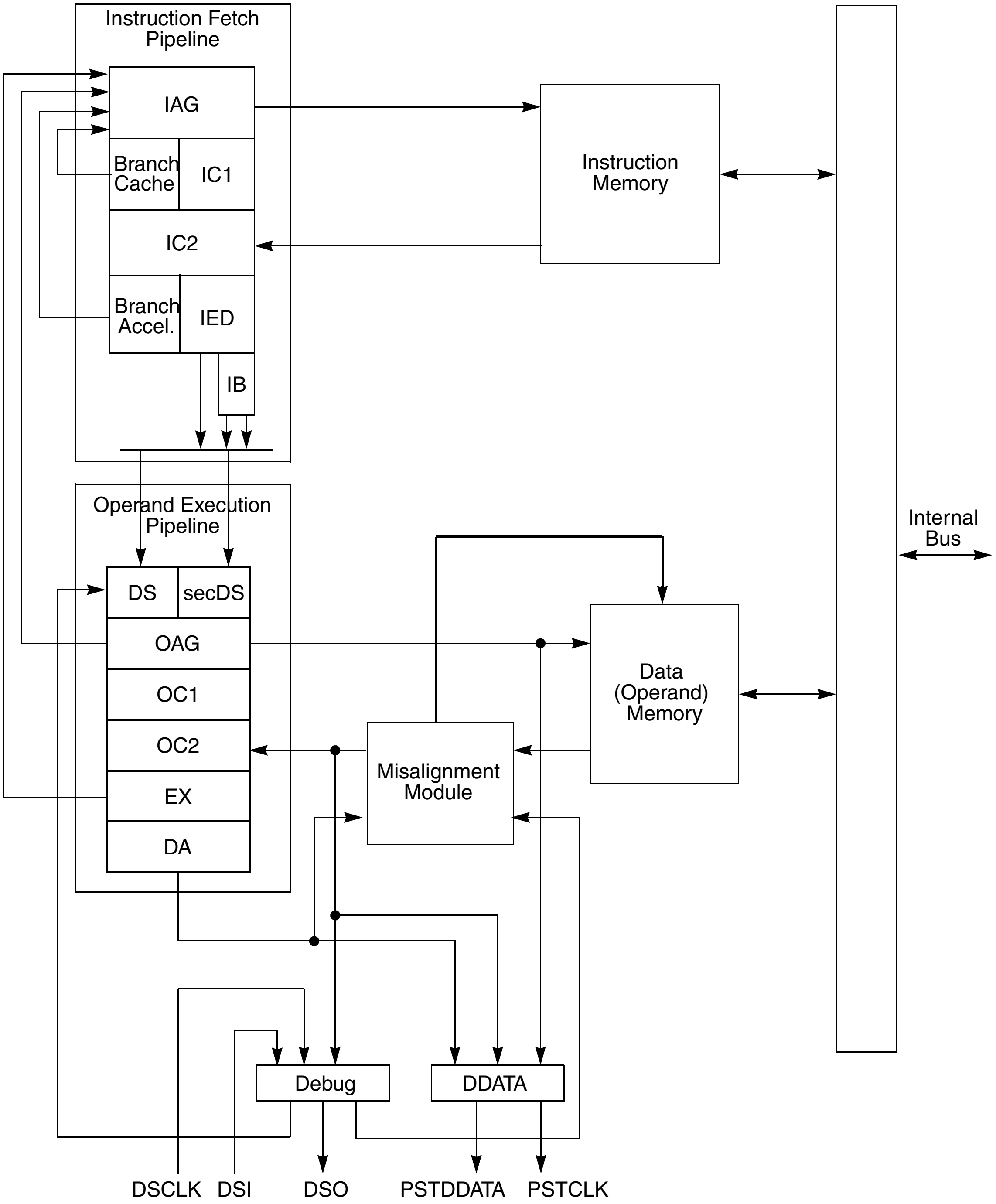


Figure 3-1. V4 ColdFire Core Pipelines

The instruction fetch pipeline (IFP) is a four-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the five-stage operand execution pipeline (OEP), that decodes the instruction, fetches the required operands, and then executes the required function. Because the IFP and OEP pipelines are decoupled by an instruction buffer serving as a FIFO queue, the IFP is able to prefetch instructions in advance of their actual use by the OEP thereby minimizing time stalled waiting for instructions.

The V4 ColdFire core pipeline stages include the following:

- Four-stage instruction fetch pipeline (IFP) (plus optional instruction buffer stage)
  - Instruction address generation (IAG) — Calculates the next prefetch address
  - Instruction fetch cycle 1 (IC1) — Prefetch on the processor's local bus
  - Instruction fetch cycle 2 (IC2) — Completes prefetch on the processor's local bus
  - Instruction early decode (IED) — Generates time-critical decode signals needed for the OEP
  - Instruction buffer (IB) — Optional buffer stage minimizes fetch latency effects using FIFO queue
- Five-stage operand execution pipeline (OEP) with two optional processor bus write cycles
  - Decode and select (DS/secDS) — Decodes and selects two sequential instructions and selects operands for effective address calculation
  - Operand address generation (OAG) — Generates the effective (logical) address
  - Operand fetch cycle 1 (OC1) — Initiates memory operand fetch on the processor's local bus
  - Operand fetch cycle 2 (OC2) — Completes memory operand fetch on the processor's local bus, as well as immediate and/or register operand fetches
  - Execute (EX) — Performs prescribed operations on previously fetched data operands
  - Write data available (DA) — Makes data available for operand write operations only
  - Store data (ST) — Updates memory element for operand write operations only

When the instruction buffer is empty, opcodes are loaded directly from the IED cycle into the operand execution pipeline. If the buffer is not empty, the IFP stores the contents of the fetched instruction and its early decode information in the IB until it is required by the OEP.

The five stage operand execution pipeline structure is a key factor in the performance of the Version 4 ColdFire design. The pipeline structure is termed a limited superscalar design because there are certain, heavily-used instruction constructs that support multiple-instruction dispatch. In particular, folding two consecutive instructions into a single pipeline issue effectively creates zero-cycle execution times for certain instructions.

With the increased performance, the bandwidth needed to support operand references requires a split bus (or Harvard architecture) where there are separate instruction and operand memory connections. These connections may be accessed concurrently to double the amount of available bandwidth to the processor's pipelines.

The resulting pipeline and local bus structure allow the V4 ColdFire core to deliver sustained high performance across a variety of demanding embedded applications.

3.1.1.1 Change-of-Flow Acceleration

To maximize the performance of conditional branch instructions, the IFP implements a sophisticated two-level acceleration mechanism. The first level is an 8-entry, direct-mapped branch cache with 2 bits for indicating four prediction states (strongly or weakly; taken or not-taken) for each entry. The branch cache also provides the association between instruction addresses and the corresponding target address. In the event of a branch cache hit, if the branch is predicted as taken, the branch cache sources the target address from the IC1 stage back into the IAG to redirect the prefetch stream to the new location as shown in Figure 3-1.

The branch cache implements instruction folding, so conditional branch instructions correctly predicted as taken can execute in zero cycles. For conditional branches with no information in the branch cache, a second-level, direct-mapped prediction table is accessed. Each of its 128 entries uses the same 2-bit prediction mechanism as the branch cache.

If a branch is predicted as taken, branch acceleration logic in the IED stage generates the target address. Other change-of-flow instructions, including unconditional branches, jumps, and subroutine calls, use a similar mechanism where the IFP calculates the target address. The performance of the subroutine return instruction (RTS) is improved through the use of a four-entry, LIFO hardware return stack. In all cases, these mechanisms allow the IFP to redirect the fetch stream down the predicted path ahead of instruction execution.

3.1.1.2 Operand Execution Pipeline (OEP)

The two instruction registers in the decode stage (DS) of the OEP are loaded from the FIFO instruction buffer or are bypassed directly from the instruction early decode (IED). The OEP consists of two traditional, two-stage RISC compute engines with a dual-ported register file access feeding an arithmetic logic unit (ALU).

The compute engine at the top of the OEP (the address ALU) is used typically for operand address calculations; the execution ALU at the bottom is used for instruction execution. The resulting structure provides almost 4 GB/s read operand bandwidth (at 250 MHz) to the two compute engines and supports single-cycle execution speeds for most instructions, including all load and store operations and most embedded-load operations. The V4 OEP supports the ColdFire instruction set architecture (ISA) revision C.

Advanced performance features implemented by the OEP:

- Stalls are minimized by dynamically basing the choice between the address ALU or execution ALU for instruction execution on the pipeline state.
- The address ALU and register renaming resources together can execute heavily used opcodes and forward results to subsequent instructions with no pipeline stalls.
- Instruction folding involving MOVE instructions allows two instructions to be issued in one cycle. The resulting microarchitecture approaches full superscalar performance at a much lower silicon cost.

3.2 Memory Map/Register Description

The following sections describe the processor registers in the user and supervisor programming models. The programming model is selected based on the processor privilege level (user mode or supervisor mode) as defined by the S bit of the status register (SR). Table 3-1 lists the processor registers.

The user-programming model consists of the following registers:

- 16 general-purpose 32-bit registers (D0–D7, A0–A7)
- 32-bit program counter (PC)

- 8-bit condition code register (CCR)
- EMAC registers (described fully in Chapter 5, “Enhanced Multiply-Accumulate Unit (EMAC
  - Four 48-bit accumulator registers partitioned as follows:
    - Four 32-bit accumulators (ACC0–ACC3)
    - Eight 8-bit accumulator extension bytes (two per accumulator). These are grouped into two 32-bit values for load and store operations (ACCEXT01 and ACCEXT23).

    Accumulators and extension bytes can be loaded, copied, and stored; results from EMAC arithmetic operations generally affect the entire 48-bit destination.
  - One 16-bit mask register (MASK)
  - One 32-bit Status register (MACSR) including four indicator bits signaling product or accumulation overflow (one for each accumulator: PAV0–PAV3)

The supervisor programming model is to be used only by system control software to implement restricted operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire processors are in the supervisor programming model, that consists of registers available in user mode as well as the following control registers:

- 16-bit status register (SR)
- 32-bit supervisor stack pointer (SSP)
- 32-bit vector base register (VBR)
- 32-bit cache control register (CACR)
- 32-bit access control registers (ACR0, ACR1, ... ACR3)
- One 32-bit memory base address register (RAMBAR)
- 32-bit address space ID register (ASID)
- 32-bit MMU base address register (MMUBAR)

Table 3-1. ColdFire Core Programming Model

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
Supervisor/User Access Registers						
Load: 0x080 Store: 0x180	Data Register 0 (D0)	32	R/W	0xCF42_602B	No	3.2.1/3-7
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x0600_2670	No	3.2.1/3-7
Load: 0x082–7 Store: 0x182–7	Data Register 2–7 (D2–D7)	32	R/W	Undefined	No	3.2.1/3-7
Load: 0x088–8E Store: 0x188–8E	Address Register 0–6 (A0–A6)	32	R/W	Undefined	No	3.2.2/3-8
Load: 0x08F Store: 0x18F	Supervisor/User A7 Stack Pointer (A7)	32	R/W	Undefined	No	3.2.3/3-8
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	No	5.2.1/5-4

Table 3-1. ColdFire Core Programming Model (continued)

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	No	5.2.2/5-6
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	No	5.2.3/5-8
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	No	5.2.4/5-8
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	No	5.2.4/5-8
0x80E	Condition Code Register (CCR)	8	R/W	Undefined	No	3.2.4/3-9
0x80F	Program Counter (PC)	32	R/W	Contents of location 0x0000_0004	No	3.2.5/3-10
Supervisor Access Only Registers						
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	3.2.6/3-10
0x003	Address Space Identifier (ASID)	8	R/W	0x00	Yes	4.2.1/4-4
0x004–7	Access Control Register 0–3 (ACR0–3)	32	R/W	See Section	Yes	6.3.2/6-8
0x008	MMU Base Address Register (MMUBAR)	32	R/W	0x0000_0000	Yes	4.2.2/4-4
0x800	User/Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x0000_0000	No	3.2.3/3-8
0x801	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes	3.2.8/3-10
0x80E	Status Register (SR)	16	R/W	0x27--	No	3.2.9/3-11
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	3.2.10/3-12

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 34, “Debug Module”.

3.2.1 Data Registers (D0–D7)

D0–D7 data registers are for bit (1-bit), byte (8-bit), word (16-bit) and longword (32-bit) operations; they can also be used as index registers.

NOTE

Registers D0 and D1 contain hardware configuration details after reset. See Section 3.3.4.15, “Reset Exception” for more details.

BDM: Load: 0x080 + *n*; *n* = 0-7 (D*n*)
Store: 0x180 + *n*; *n* = 0-7 (D*n*)

Access: User read/write
BDM 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
R W	Data							
Reset (D2-D7)	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
Reset (D0, D1)	See Section 3.3.4.15, “Reset Exception”							

Figure 3-2. Data Registers (D0–D7)

3.2.2 Address Registers (A0–A6)

These registers can be used as software stack pointers, index registers, or base address registers. They can also be used for word and longword operations.

BDM: Load: 0x088 + *n*; *n* = 0–6 (A*n*)
Store: 0x188 + *n*; *n* = 0–6 (A*n*)

Access: User read/write
BDM 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
R W	Address							
Reset	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –

Figure 3-3. Address Registers (A0–A6)

3.2.3 Supervisor/User Stack Pointers (A7 and OTHER_A7)

The ColdFire architecture supports two independent stack pointer (A7) registers—the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two program-visible 32-bit registers does not identify one as the SSP and the other as the USP. Instead, the hardware uses one 32-bit register as the active A7 and the other as OTHER_A7. Thus, the register contents are a function of the processor operation mode, as shown in the following:

```
if SR[S] = 1
         then     A7 = Supervisor Stack Pointer
                  OTHER_A7 = User Stack Pointer
         else     A7 = User Stack Pointer
                  OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports direct reads and writes to the (active) A7 and OTHER_A7. It is the responsibility of the external development system to determine, based on the setting of SR[S], the mapping of A7 and OTHER_A7 to the two program-visible definitions (SSP and USP). This functionality is enabled by setting the enable user stack pointer bit, CACR[EUSP]. If this bit is cleared, only a single stack pointer (A7), originally defined for ColdFire ISA_A, is available. EUSP is cleared at reset.

To support dual stack pointers, the following two supervisor instructions are included in the ColdFire instruction set architecture to load/store the USP:

```
      move.l Ay,USP;move to USP
```

```
    move.l USP,Ax;move from USP
```

These instructions are described in the *ColdFire Family Programmer’s Reference Manual*. All other instruction references to the stack pointer, explicit or implicit, access the active A7 register.

NOTE

The SSP is loaded during reset exception processing with the contents of location 0x0000_0000.

BDM: Load: 0x08F (A7)
Store: 0x18F (A7)
0x800 (OTHER_A7)

Access: A7: User or BDM read/write
OTHER_A7: Supervisor or BDM 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
R W	Address																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 3-4. Stack Pointer Registers (A7 and OTHER_A7)

3.2.4 Condition Code Register (CCR)

The CCR is the LSB of the processor status register (SR). Bits 4–0 act as indicator flags for results generated by processor operations. The extend bit (X) is also an input operand during multiprecision arithmetic computations.

NOTE

The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.

BDM: LSB of Status Register (SR)

Access: User read/write
BDM read/write

	7	6	5	4	3	2	1	0
R	0	0	0	X	N	Z	V	C
W								
Reset:	0	0	0	—	—	—	—	—

Figure 3-5. Condition Code Register (CCR)

Table 3-2. CCR Field Descriptions

Field	Description
7–5	Reserved, must be cleared.
4 X	Extend condition code bit. Set to the C-bit value for arithmetic operations; otherwise not affected or set to a specified result.
3 N	Negative condition code bit. Set if most significant bit of the result is set; otherwise cleared.

Table 3-2. CCR Field Descriptions (continued)

Field	Description
2 Z	Zero condition code bit. Set if result equals zero; otherwise cleared.
1 V	Overflow condition code bit. Set if an arithmetic overflow occurs implying the result cannot be represented in operand size; otherwise cleared.
0 C	Carry condition code bit. Set if a carry out of the operand msb occurs for an addition or if a borrow occurs in a subtraction; otherwise cleared.

3.2.5 Program Counter (PC)

The PC contains the currently executing instruction address. During instruction execution and exception processing, the processor automatically increments PC contents or places a new value in the PC. The PC is a base address for PC-relative operand addressing.

The PC is initially loaded during reset exception processing with the contents at location 0x0000_0004.

BDM: 0x80F (PC)

Access: User read/write
BDM 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
R	Address																															
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 3-6. Program Counter Register (PC)

3.2.6 Cache Programming Model

The registers in the cache portion of the programming model are described in Chapter 6, “Cache.”

3.2.7 MMU Programming Model

The registers in the MMU portion of the programming model are described in Chapter 4, “Memory Management Unit (MMU).”

3.2.8 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in the memory. To access the vector table, the displacement of an exception vector is added to the value in VBR. The lower 20 bits of the VBR are not implemented by ColdFire processors. They are assumed to be zero, forcing the table to be aligned on a 1 MB boundary.

BDM: 0x801 (VBR)

Access: Supervisor read/write
BDM 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
R	Base Address												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-7. Vector Base Register (VBR)

3.2.9 Status Register (SR)

The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits (CCR) are accessible. The control bits indicate the following states for the processor: trace mode (T bit), supervisor or user mode (S bit), and master or interrupt state (M bit). All defined bits in the SR have read/write access when in supervisor mode.

NOTE

The lower byte of the SR (the CCR) must be loaded explicitly after reset and before any compare (CMP), Bcc, or Scc instructions execute.

BDM: 0x80E (SR)

Access: Supervisor read/write
BDM read/write

	System Byte								Condition Code Register (CCR)							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	T	0	S	M	0	I			0	0	0	X	N	Z	V	C
W																
Reset	0	0	1	0	0	1	1	1	0	0	0	—	—	—	—	—

Figure 3-8. Status Register (SR)

Table 3-3. SR Field Descriptions

Field	Description
15 T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	Reserved, must be cleared.
13 S	Supervisor/user state. 0 User mode 1 Supervisor mode
12 M	Master/interrupt state. Bit is cleared by an interrupt exception and software can set it during execution of the RTE or move to SR instructions.
11	Reserved, must be cleared.

Table 3-3. SR Field Descriptions (continued)

Field	Description
10–8 I	Interrupt level mask. Defines current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to current level, except edge-sensitive level 7 requests, which cannot be masked.
7–0 CCR	Refer to Section 3.2.4, “Condition Code Register (CCR)”.

3.2.10 Memory Base Address Register (RAMBAR)

The memory base address register is used to specify the base address of the internal SRAM module and indicates the types of references mapped to it. The base address register includes a base address, write-protect bit, address space mask bits, and an enable bit. RAMBAR determines the base address of the on-chip RAM. For more information, refer to Section 7.2.1, “SRAM Base Address Register (RAMBAR)”.

3.3 Functional Description

3.3.1 Version 4 ColdFire Microarchitecture

As previously discussed, the unrolling of the operand execution pipeline into a five-stage structure is a key factor in the improved performance of the Version 4 ColdFire design. The resulting pipeline structure is termed a limited superscalar design because there are certain, heavily-used instruction constructs that support multiple-instruction dispatch. The following figure presents the top-level spatial block diagram of the Version 4 ColdFire operand execution pipeline, where the major hardware structures associated with each pipeline stage are clearly visible.

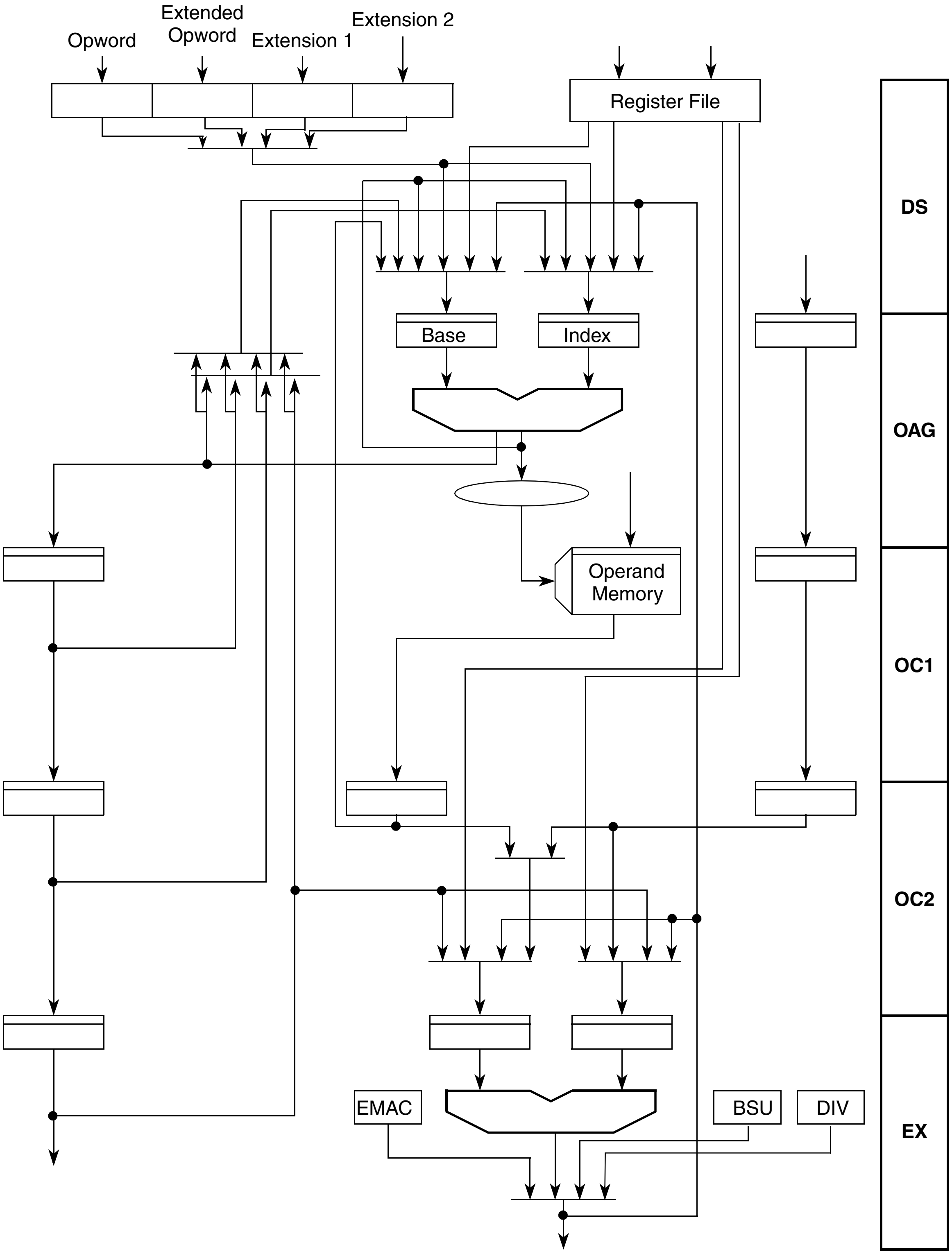


Figure 3-9. Version 4 ColdFire Processor Operand Execution Pipeline Diagram

3.3.2 Instruction Set Architecture (ISA_C)

The original ColdFire instruction set architecture (ISA_A) was derived from the M68000 family opcodes based on extensive analysis of embedded application code. The ISA was optimized for code compiled from high-level languages where the dominant operand size was the 32-bit integer declaration. This approach minimized processor complexity and cost, while providing excellent performance for compiled applications.

After the initial ColdFire compilers were created, developers noted there were certain ISA additions that would enhance code density and overall performance. Additionally, as users implemented ColdFire-based designs into a wide range of embedded systems, they found certain frequently-used instruction sequences that could be improved by the creation of additional instructions.

The original ISA definition minimized support for instructions referencing byte- and word-sized operands. Full support for the move byte and move word instructions was provided, but the only other opcodes supporting these data types are CLR (clear) and TST (test). A set of instruction enhancements has been implemented in subsequent ISA revisions, ISA_B and ISA_C. The added opcodes primarily addressed three areas:

1. Enhanced support for byte and word-sized operands
2. Enhanced support for position-independent code
3. Miscellaneous instruction additions to address new functionality

Table 3-4 summarizes the instructions added to revision ISA_A to form revision ISA_C. For more details see the *ColdFire Family Programmer's Reference Manual*.

Table 3-4. Instruction Enhancements over Revision ISA_A

Instruction	Description
BITREV	The contents of the destination data register are bit-reversed; that is, new Dn[31] equals old Dn[0], new Dn[30] equals old Dn[1], ..., new Dn[0] equals old Dn[31].
BYTEREV	The contents of the destination data register are byte-reversed; that is, new Dn[31:24] equals old Dn[7:0], ..., new Dn[7:0] equals old Dn[31:24].
FF1	The data register, Dn, is scanned, beginning from the most-significant bit (Dn[31]) and ending with the least-significant bit (Dn[0]), searching for the first set bit. The data register is then loaded with the offset count from bit 31 where the first set bit appears.
INTOUCH	Loads blocks of instructions to be locked in the instruction cache.
MOV3Q.L	Moves 3-bit immediate data to the destination location.
Move from USP	User Stack Pointer → Destination register
Move to USP	Source register → User Stack Pointer
MVS.{B,W}	Sign-extends source operand and moves it to destination register.
MVZ.{B,W}	Zero-fills source operand and moves it to destination register.
SATS.L	Performs saturation operation for signed arithmetic and updates destination register, depending on CCR[V] and bit 31 of the register.
TAS.B	Performs indivisible read-modify-write cycle to test and set addressed memory byte.
Bcc.L	Branch conditionally, longword

Table 3-4. Instruction Enhancements over Revision ISA_A (continued)

Instruction	Description
BSR.L	Branch to sub-routine, longword
CMP.{B,W}	Compare, byte and word
CMPA.W	Compare address, word
CMPI.{B,W}	Compare immediate, byte and word
MOVEI	Move immediate, byte and word to memory using Ax with displacement

3.3.3 Exception Processing Overview

Exception processing for ColdFire processors is streamlined for performance. The ColdFire processors differ from the M68000 family because they include:

- A simplified exception vector table
- Reduced relocation capabilities using the vector-base register
- A single exception stack frame format
- A precise instruction restart model for translation (TLB miss) and access faults. This functionality extends the existing ColdFire access error fault vector in the exception stack frames.

All ColdFire processors use an instruction restart exception model. Exception processing includes all actions from fault condition detection to the initiation of fetch for first handler instruction. Exception processing is comprised of four major steps:

1. The processor makes an internal copy of the SR and then enters supervisor mode by setting the S bit and disabling trace mode by clearing the T bit. The interrupt exception also forces the M bit to be cleared and the interrupt priority mask to set to current interrupt request level.

2. The processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from the interrupt controller. The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address.

3. The processor saves the current context by creating an exception stack frame on the system stack. The exception stack frame is created at a 0-modulo-4 address on top of the system stack pointed to by the supervisor stack pointer (SSP). As shown in Figure 3-10, the processor uses a simplified fixed-length stack frame for all exceptions with additional fault status (FS) encodings to support the MMU. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction (fault) or the address of the next instruction to be executed (next).
4. The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 MB boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register.

The index into the exception table is calculated as (4 × vector number). After the exception vector has been fetched, the vector contents determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has initiated, exception processing terminates and normal instruction processing continues in the handler.

All ColdFire processors support a 1024-byte vector table aligned on any 1 Mbyte address boundary (see Table 3-5).

The table contains 256 exception vectors; the first 64 are defined for the core and the remaining 192 are device-specific peripheral interrupt vectors. See Chapter 17, “Interrupt Controller Modules” for details on the device-specific interrupt sources.

Table 3-5. Exception Vector Assignments

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5	0x014	Fault	Divide by zero
6–7	0x018–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented line-F opcode
12	0x030	Next	Non-PC breakpoint debug interrupt
13	0x034	Next	PC breakpoint debug interrupt
14	0x038	Fault	Format error
15	0x03C	Next	Uninitialized interrupt
16–23	0x040–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064–0x07C	Next	Level 1–7 autovectored interrupts
32–47	0x080–0x0BC	Next	Trap # 0-15 instructions
48–60	0x0C0–0x0F0	—	Reserved
61	0x0F4	Fault	Unsupported instruction
62–63	0x0F8–0x0FC	—	Reserved
64–255	0x100–0x3FC	Next	Device-specific interrupts

[1] Fault refers to the PC of the instruction that caused the exception. Next refers to the PC of the instruction that follows the instruction that caused the fault.

All ColdFire processors inhibit interrupt sampling during the first instruction of all exception handlers. This allows any handler to disable interrupts effectively, if necessary, by raising the interrupt mask level contained in the status register. For more details, see *ColdFire Family Programmer's Reference Manual*.

3.3.3.1 Exception Stack Frame Definition

Figure 3-10 shows exception stack frame. The first longword contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second longword contains the 32-bit program counter address.

	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
SSP →	Format	FS[3:2]	Vector	FS[1:0]	Status Register
+ 0x4	Program Counter				

Figure 3-10. Exception Stack Frame Form

The 16-bit format/vector word contains three unique fields:

- A 4-bit format field at the top of the system stack is always written with a value of 4, 5, 6, or 7 by the processor, indicating a two-longword frame format. See Table 3-6.

Table 3-6. Format Field Encodings

Original SSP @ Time of Exception, Bits 1:0	SSP @ 1st Instruction of Handler	Format Field
00	Original SSP - 8	0100
01	Original SSP - 9	0101
10	Original SSP - 10	0110
11	Original SSP - 11	0111

- There is a 4-bit fault status field, FS[3:0], at the top of the system stack. This field is defined for access and address errors only and written as zeros for all other exceptions. See Table 3-7.

Table 3-7. Fault Status Encodings

FS[3:0]	Definition
0000	Not an access or address error nor an interrupted debug service routine
0001	Reserved
0010	Interrupt during a debug service routine for faults other than access errors[1]
0011	Reserved
0100	Error on instruction fetch
0101	TLB miss on opword of instruction fetch
0110	TLB miss on extension word of instruction fetch
0111	IFP access error while executing in emulator mode
1000	Error on operand write
1001	Attempted write to write-protected space

Table 3-7. Fault Status Encodings (continued)

FS[3:0]	Definition
1010	TLB miss on data write
1011	Reserved
1100	Error on operand read
1101	Attempted read, read-modify-write of protected space
1110	TLB miss on data read, or read-modify-write
1111	OEP access error while executing in emulator mode

[1] This refers to taking an I/O interrupt during a debug service routine. If an access error occurs during a debug service routine, FS is set to 0111 if it is due to an instruction fetch or to 1111 for a data access.

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the interrupt controller in case of an interrupt. See Table 3-5.

3.3.4 Processor Exceptions

3.3.4.1 Access Error Exception

The exact processor response to an access error depends on the memory reference being performed. For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults during instruction prefetches followed by a change of instruction flow do not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error is signaled and the instruction is aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. The operand execution pipeline includes logic to fully recover program-visible register updates in the event of a bus transfer error acknowledge on an operand memory reference. This allows for a precise instruction restart from this class of exceptions. See Section 3.3.4.16, "Precise Faults", for additional information.

If the MMU is disabled, access errors are reported only with an attempted store to write-protected memory. Therefore, access errors associated with instruction fetch or operand read accesses are not possible. The Version 4 ColdFire processor, unlike the Version 2 and 3 ColdFire processors, updates the condition code register if a write-protect error occurs during a CLR or MOV3Q operation to memory.

Internal memory accesses that fault (terminate with an internal memory transfer error acknowledge) generate an access error exception. MMU TLB misses and access violations use the same fault. If the MMU is enabled, all TLB misses and protection violations generate an access error exception. To determine if a fault is due to a TLB miss or another type of access error, new FS encodings (described in Table 3-7) signal TLB misses on instruction fetch, instruction extension fetch, and data read and writes.

3.3.4.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (if bit 0 of the target address is set) results in an address error exception.

Any attempted use of a word-sized index register (Xn.w) or a scale factor of eight on an indexed effective addressing mode generates an address error, as does an attempted execution of a full-format indexed addressing mode, which is defined by bit 8 of extension word 1 being set.

If an address error occurs on a JSR instruction, the Version 4 ColdFire processor first pushes the return address onto the stack and then calculates the target address. If an address error occurs on an RTS instruction, the Version 4 ColdFire processor preserves the original return PC and writes the exception stack frame above this value.

3.3.4.3 Illegal Instruction Exception

The ColdFire variable-length instruction set architecture supports three instruction sizes: 16, 32, or 48 bits. The first instruction word is known as the operation word (or opword), while the optional words are known as extension word 1 and extension word 2. The opword is further subdivided into three sections: the upper four bits segment the entire ISA into 16 instruction lines, the next 6 bits define the operation mode (opmode), and the low-order 6 bits define the effective address. See Figure 3-11. The opword line definition is shown in Table 3-8.

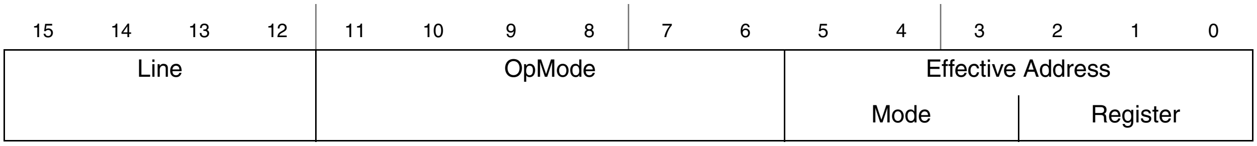


Figure 3-11. ColdFire Instruction Operation Word (Opword) Format

Table 3-8. ColdFire Opword Line Definition

Opword[Line]	Instruction Class
0x0	Bit manipulation, Arithmetic and Logical Immediate
0x1	Move Byte
0x2	Move Long
0x3	Move Word
0x4	Miscellaneous
0x5	Add (ADDQ) and Subtract Quick (SUBQ), Set according to Condition Codes (Scc)
0x6	PC-relative change-of-flow instructions Conditional (Bcc) and unconditional (BRA) branches, subroutine calls (BSR)
0x7	Move Quick (MOVEQ), Move with sign extension (MVS) and zero fill (MVZ)
0x8	Logical OR (OR)
0x9	Subtract (SUB), Subtract Extended (SUBX)
0xA	EMAC, Move 3-bit Quick (MOV3Q)
0xB	Compare (CMP), Exclusive-OR (EOR)

Table 3-8. ColdFire Opword Line Definition (continued)

Opword[Line]	Instruction Class
0xC	Logical AND (AND), Multiply Word (MUL)
0xD	Add (ADD), Add Extended (ADDX)
0xE	Arithmetic and logical shifts (ASL, ASR, LSL, LSR)
0xF	Cache Push (CPUSHL), Write DDATA (WDDATA), Write Debug (WDEBUG)

In the original M68000 ISA definition, lines A and F were effectively reserved for user-defined operations (line A) and co-processor instructions (line F). Accordingly, there are two unique exception vectors associated with illegal opwords in these two lines.

Any attempted execution of an illegal 16-bit opcode (except for line-A and line-F opcodes) generates an illegal instruction exception (vector 4). Additionally, any attempted execution of any non-MAC line-A and most line-F opcodes generate their unique exception types, vector numbers 10 and 11, respectively. ColdFire cores do not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

3.3.4.4 Divide-By-Zero

Attempting to divide by zero causes an exception (vector 5, offset equal 0x014).

3.3.4.5 Privilege Violation

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception. See *ColdFire Programmer's Reference Manual* for a list of supervisor-mode instructions.

There is one special case involving the HALT instruction. Normally, this opcode is a supervisor mode instruction, but if the debug module's CSR[UHE] is set, then this instruction can be also be executed in user mode for debugging purposes.

3.3.4.6 Trace Exception

To aid in program development, all ColdFire processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by setting of the SR[T] bit, the completion of an instruction execution (for all but the stop instruction) signals a trace exception. This functionality allows a debugger to monitor program execution.

The stop instruction has the following effects:

1. The instruction before the stop executes and then generates a trace exception. In the exception stack frame, the PC points to the stop opcode.
2. When the trace handler is exited, the stop instruction executes, loading the SR with the immediate operand from the instruction.
3. The processor then generates a trace exception. The PC in the exception stack frame points to the instruction after the stop, and the SR reflects the value loaded in the previous step.

If the processor is not in trace mode and executes a stop instruction where the immediate operand sets SR[T], hardware loads the SR and generates a trace exception. The PC in the exception stack frame points to the instruction after the stop, and the SR reflects the value loaded in step 2.

Because ColdFire processors do not support any hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider a TRAP instruction execution while in trace mode. The processor initiates the trap exception and then passes control to the corresponding handler. If the system requires that a trace exception be processed, it is the responsibility of the trap exception handler to check for this condition (SR[T] in the exception stack frame set) and pass control to the trace handler before returning from the original exception.

3.3.4.7 Unimplemented Line-A Opcode

A line-A opcode is defined when bits 15-12 of the opword are 0b1010. This exception is generated by the attempted execution of an undefined line-A opcode.

3.3.4.8 Unimplemented Line-F Opcode

A line-F opcode is defined when bits 15-12 of the opword are 0b1111. This exception is generated when attempting to execute an undefined line-F opcode.

3.3.4.9 Debug Interrupts

See Chapter 34, “Debug Module,” for a detailed explanation of these exceptions, which are generated in response to hardware breakpoint register triggers. The processor does not generate an IACK cycle, but rather calculates the vector number internally (vector number 12 or 13, depending on the type of breakpoint trigger). Additionally, SR[M,I] are unaffected by the interrupt.

Separate exception vectors are provided for PC breakpoints and for address/data breakpoints. In the case of a two-level trigger, the last breakpoint determines the vector. There are two unique vectors for these exceptions: vector 0x030 corresponds to non-PC breakpoints and vector 0x034 corresponds to PC breakpoints.

3.3.4.10 RTE and Format Error Exception

When an RTE instruction is executed, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire core, any attempted RTE execution (where the format is not equal to {4,5,6,7}) generates a format error. The exception stack frame for the format error is created without disturbing the original RTE frame and the stacked PC pointing to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from M68000 applications. On M68000 family processors, the SR was located at the top of the stack. On those processors, bit 30 of the longword addressed by the system stack pointer is typically zero. Thus, if an RTE is attempted using this old format, it generates a format error on a ColdFire processor.

If the format field defines a valid type, the processor: (1) reloads the SR operand, (2) fetches the second longword operand, (3) adjusts the stack pointer by adding the format value to the auto-incremented address after the fetch of the first longword, and then (4) transfers control to the instruction address defined by the second longword operand within the stack frame.

3.3.4.11 TRAP Instruction Exception

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls. The TRAP instruction may be used to change from user to supervisor mode.

3.3.4.12 Unsupported Instruction Exception

If execution of a valid instruction is attempted but the required hardware is not present in the processor, an unsupported instruction exception is generated. The instruction functionality can then be emulated in the exception handler, if desired.

All ColdFire cores record the processor hardware configuration in the D0 register immediately after the negation of $\overline{\text{RESET}}$. See Section 3.3.4.15, “Reset Exception,” for details.

3.3.4.13 Interrupt Exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle. See Chapter 17, “Interrupt Controller Modules,” for details on the interrupt controller.

3.3.4.14 Fault-on-Fault Halt

If a ColdFire processor encounters any type of fault during the exception processing of another fault, the processor immediately halts execution with the catastrophic fault-on-fault condition. A reset is required to to exit this state.

3.3.4.15 Reset Exception

Asserting the reset input signal ($\overline{\text{RESET}}$) to the processor causes a reset exception. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. Reset also aborts any processing in progress when the reset input is recognized. Processing cannot be recovered.

The reset exception places the processor in the supervisor mode by setting the SR[S] bit and disables tracing by clearing the SR[T] bit. This exception also clears the SR[M] bit and sets the processor’s SR[I] field to the highest level (level 7, 0b111). Next, the VBR is initialized to zero (0x0000_0000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

NOTE

> Other implementation-specific registers are also affected. Refer to each module in this reference manual for details on these registers.

After the processor is granted the bus, it performs two longword read-bus cycles. The first longword at address 0x0000_0000 is loaded into the supervisor stack pointer and the second longword at address

0x0000_0004 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault state.

ColdFire processors load hardware configuration information into the D0 and D1 general-purpose registers after system reset. The hardware configuration information is loaded immediately after the reset-in signal is negated. This allows an emulator to read out the contents of these registers via the BDM to determine the hardware configuration.

Information loaded into D0 defines the processor hardware configuration as shown in Figure 3-12.

BDM: Load: 0x080 (D0)
Store: 0x180 (D0)

Access: User read-only
BDM read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PF								VER				REV			
W																
Reset	1	1	0	0	1	1	1	1	0	1	0	0	0	0	1	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAC	DIV	EMAC	FPU	0	0	0	0	ISA				DEBUG			
W																
Reset	0	1	1	0	0	0	0	0	0	0	1	0	1	0	1	1

Figure 3-12. D0 Hardware Configuration Info

Table 3-9. D0 Hardware Configuration Info Field Description

Field	Description
31–24 PF	Processor family. This field is fixed to a hex value of 0xCF indicating a ColdFire core is present.
23–20 VER	ColdFire core version number. Defines the hardware microarchitecture version of ColdFire core. 0001 V1 ColdFire core 0010 V2 ColdFire core 0011 V3 ColdFire core 0100 V4 ColdFire core (This is the value used for this device.) 0101 V5 ColdFire core Else Reserved for future use
19–16 REV	Processor revision number. The default is 0b0010.
15 MAC	MAC present. This bit signals if the optional multiply-accumulate (MAC) execution engine is present in processor core. 0 MAC execute engine not present in core. (This is the value used for this device.) 1 MAC execute engine is present in core.
14 DIV	Divide present. This bit signals if the hardware divider (DIV) is present in the processor core. 0 Divide execute engine not present in core. 1 Divide execute engine is present in core. (This is the value used for this device.)
13 EMAC	EMAC present. This bit signals if the optional enhanced multiply-accumulate (EMAC) execution engine is present in processor core. 0 EMAC execute engine not present in core. 1 EMAC execute engine is present in core. (This is the value used for this device.)

Table 3-9. D0 Hardware Configuration Info Field Description (continued)

Field	Description
12 FPU	FPU present. This bit signals if the optional floating-point (FPU) execution engine is present in processor core. 0 FPU execute engine not present in core. (This is the value used for this device.) 1 FPU execute engine is present in core.
11–8	Reserved.
7–4 ISA	ISA revision. Defines the instruction-set architecture (ISA) revision level implemented in ColdFire processor core. 0000 ISA_A 0001 ISA_B 0010 ISA_C (This is the value used for this device.) 1000 ISA_A+ Else Reserved
3–0 DEBUG	Debug module revision number. Defines revision level of the debug module used in the ColdFire processor core. 0000 DEBUG_A 0001 DEBUG_B 0010 DEBUG_C 0011 DEBUG_D 0100 DEBUG_E 1001 DEBUG_B+ 1011 DEBUG_D+ (This is the value used for this device.) 1111 DEBUG_D+PST Buffer Else Reserved

Information loaded into D1 defines the local memory hardware configuration as shown in the figure below.

BDM: Load: 0x081 (D1)
Store: 0x181 (D1)

Access: User read-only
BDM read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CLSZ		ICAS		ICSZ				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MBSZ		CPES	DCAS	DCSZ				SRAMSZ					0	0	0
W																
Reset	0	0	1	0	0	1	1	0	0	1	1	1	0	0	0	0

Figure 3-13. D1 Hardware Configuration Info

Table 3-10. D1 Hardware Configuration Information Field Description

Field	Description
31–30 CLSZ	Cache line size. This field is fixed to a hex value of 0x0 indicating a 16-byte cache line size.
29–28 ICAS	Instruction cache associativity. 00 Four-way (This is the value used for this device) 01 Direct mapped Else Reserved for future use
27–24 ICSZ	Instruction cache size. Indicates the amount of instruction cache. 0000 No instruction cache 0001 512 B instruction cache 0010 1 KB instruction cache 0011 2 KB instruction cache 0100 4 KB instruction cache 0101 8 KB instruction cache 0110 16 KB instruction cache (This is the value used for this device) 0111 32 KB instruction cache 1000 64 KB instruction cache Else Reserved
23–16	Reserved.
15–14 MBSZ	Bus size. Defines the width of the ColdFire master bus datapath. 00 32-bit system bus datapath (This is the value used for this device) 01 64-bit system bus datapath Else Reserved
13 CPES	CPUSHL enhancements supported. Specifies whether the enhancements to the CPUSHL instructions are supported by the processor core. See Section 6.4.8, “CPUSHL Enhancements,” for details. 0 CPUSHL instruction enhancements are not supported 1 CPUSHL instruction enhancements are supported (This is the value used for this device)
12 DCAS	Data cache associativity. Defines the data cache set-associativity. 0 Four-way (This is the value used for this device) 1 Direct mapped
11–8 DCSZ	Data cache size. Indicates the size of the unified cache. 0000 No data cache 0001 512 bytes 0010 1 KB 0011 2 KB 0100 4 KB 0101 8 KB 0110 16 KB (This is the value used for this device) 0111 32 KB Else Reserved for future use

Table 3-10. D1 Hardware Configuration Information Field Description (continued)

Field	Description
7–3 SRAMSZ	SRAM bank size. 00000 No SRAM 00010 512 bytes 00100 1 KB 00110 2 KB 01000 4 KB 01010 8 KB 01100 16 KB 01111 24 KB 01110 32 KB (This is the value used for this device) 10000 64 KB 10010 128 KB Else Reserved for future use
2–0	Reserved.

3.3.4.16 Precise Faults

To support a demand-paged virtual-memory environment, all memory references require precise, recoverable faults. The ColdFire instruction restart mechanism ensures that a faulted instruction restarts from the execution beginning. No internal state information is saved when an exception occurs nor is any restored when the handler ends. Given the PC address defined in the exception stack frame, the processor re-establishes program execution by transferring control to the given location as part of the RTE (return from exception) instruction.

The instruction restart recovery model requires program-visible register changes made during execution to be undone if that instruction subsequently faults.

The Version 4 (and later) ColdFire OEP structure naturally supports this concept for most instructions; program-visible registers are updated only in the final OEP stage when fault collection is complete. If any exception occurs, pending register updates are discarded.

For V4 ColdFire cores and later, most single-cycle instructions naturally support precise faults and instruction restart, while complex instruction do not. Consider the following memory-to-memory move:

```
        move.l   (Ay)+,(Ax)+        # copy 4 bytes from source to destination
```

This instruction takes one cycle to read the source operand (Ay) and one to write the data into Ax. Source and destination address pointers are updated as part of execution. Table 3-11 lists the operations performed in execute stage (EX).

Table 3-11. OEP EX Cycle Operations

EX Cycle	Operations
1	Read source operand from memory @ (Ay), update Ay, new Ay = old Ay + 4
2	Write operand into destination memory @ (Ax), update Ax, new Ax = old Ax + 4, update CCR

A fault detected with the destination memory write is reported during the second cycle. At this point, operations performed in the first cycle are complete, so if the destination write takes any type of access error, Ay is updated. After the access error handler executes and the faulting instruction restarts, the processor's operation would be incorrect (without the special register recovery hardware) because the source-address register has an incorrect (post-incremented) value.

To recover the original state of the programming model for all instructions, the Version 4 ColdFire core adds the needed hardware to support full-register recovery. This hardware allows program-visible registers to be restored to their original state for multi-cycle instructions so that the instruction restart mechanism is supported. Memory-to-memory moves and move-multiple loads are representative of the complex instructions needing the special recovery support.

Recall the IFP and OEP are decoupled by a FIFO instruction buffer. In the V4 ColdFire IFP, each buffer entry includes 48 bits of instruction data fetched from memory and 64 bits of early decode and branch prediction information. This datapath also includes IFP fault-status information. Therefore, every IFP access can be tagged if an instruction fetch terminates with an error acknowledge. IFP access errors are recognized after the buffered instruction enters the OEP.

NOTE

For access errors signaled on instruction prefetches, an access error exception is generated only if instruction execution is attempted. If an instruction fetch access error exception is generated and the FS field indicates the fault occurred on an extension word, it may be necessary for the exception PC to be rounded-up to the next page address to determine the faulting instruction fetch address.

3.3.5 Instruction Execution Timing

This section presents processor instruction execution times in terms of processor-core clock cycles. The number of operand references for each instruction is enclosed in parentheses following the number of processor clock cycles. Each timing entry is presented as C(R/W) where:

- C is the number of processor clock cycles, including all applicable operand fetches and writes, and all internal core cycles required to complete the instruction execution.
- R/W is the number of operand reads (R) and writes (W) required by the instruction. An operation performing a read-modify-write function is denoted as (1/1).

This section includes the assumptions concerning the timing values and the execution time details.

3.3.5.1 Timing Assumptions

For the timing data presented in this section, these assumptions apply:

1. The OEP is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP does not wait for the IFP to supply opwords and/or extension words.
2. Execution times for individual instructions make no assumptions concerning the OEP's ability to dispatch multiple instructions in one machine cycle. For sequences where instruction pairs are issued, the execution time of the first instruction defines the execution time of pair; the second instruction effectively executes in zero cycles.
3. The OEP does not experience any sequence-related pipeline stalls. The most common example of stall occurs when a register is modified in the EX engine and a subsequent instruction generates an address that uses the previously modified register. The second instruction stalls in the OEP until the previous instruction updates the register. For example, in the following code:

```
muls.l   #<data>,d0
move.l   (a0,d0.l*4),d1
```

the move.l instruction waits three cycles for the muls.l to update D0. If consecutive instructions update a register and use that register as a base of index value with a scale factor of 1 (Xi.l*1) in an address calculation, a 2-cycle pipeline stall occurs. If the destination register is used as an index register with any other scale factor (Xi.l*2, Xi.l*4), a 3-cycle stall occurs.

NOTE

> Address register results from post-increment and pre-decrement modes are available to subsequent instructions without stalls.

4. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume that an infinite zero-wait state memory is attached to the processor core.
5. All operand data accesses are aligned on the same byte boundary as the operand size; for example, 16-bit operands aligned on 0-modulo-2 addresses, 32-bit operands aligned on 0-modulo-4 addresses.

The processor core decomposes misaligned operand references into a series of aligned accesses as shown in Table 3-12.

Table 3-12. Misaligned Operand References

address[1:0]	Size	Bus Operations	Additional C(R/W)
01 or 11	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
01 or 11	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

3.3.5.2 MOVE Instruction Execution Times

Table 3-13 lists execution times for MOVE.{B,W} instructions; Table 3-14 lists timings for MOVE.L.

NOTE

For all tables in this section, the execution time of any instruction using the PC-relative effective addressing modes is the same for the comparable An-relative mode.

ET with {<ea> = (d16,PC)}	equals ET with {<ea> = (d16,An)}
ET with {<ea> = (d8,PC,Xi*SF)}	equals ET with {<ea> = (d8,An,Xi*SF)}

The nomenclature xxx.wl refers to both forms of absolute addressing, xxx.w and xxx.l.

Table 3-13. MOVE Byte and Word Execution Times

Source	Destination						
	Rx	**(Ax)**	**(Ax)+**	**-(Ax)**	**(d16,Ax)**	**(d8,Ax,Xi*SF)**	**xxx.wl**
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1))	2(1/1)
(Ay)+	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1))	2(1/1)
-(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1))	2(1/1)
(d16,Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,Ay,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1))	—	—	—
#xxx	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	—	—

Table 3-14. MOVE Long Execution Times

Source	Destination						
	Rx	**(Ax)**	**(Ax)+**	**-(Ax)**	**(d16,Ax)**	**(d8,Ax,Xi*SF)**	**xxx.wl**
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—

Table 3-14. MOVE Long Execution Times (continued)

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
(d8,Ay,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#xxx	1(0/0)	1(0/1)	1(0/1)	1(0/1)	—	—	—

3.3.5.3 Standard One Operand Instruction Execution Times

Table 3-15. One Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
BITREV	Dx	1(0/0)	—	—	—	—	—	—	—
BYTEREV	Dx	1(0/0)	—	—	—	—	—	—	—
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.L	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
EXT.W	Dx	1(0/0)	—	—	—	—	—	—	—
EXT.L	Dx	1(0/0)	—	—	—	—	—	—	—
EXTB.L	Dx	1(0/0)	—	—	—	—	—	—	—
FF1	Dx	1(0/0)	—	—	—	—	—	—	—
NEG.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEGX.L	Dx	1(0/0)	—	—	—	—	—	—	—
NOT.L	Dx	1(0/0)	—	—	—	—	—	—	—
SATS.L	Dx	1(0/0)	—	—	—	—	—	—	—
SCC	Dx	1(0/0)	—	—	—	—	—	—	—
SWAP	Dx	1(0/0)	—	—	—	—	—	—	—
TAS.B	<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
TST.B	<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
TST.W	<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
TST.L	<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)

3.3.5.4 Standard Two Operand Instruction Execution Times

Table 3-16. Two Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
ADD.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
ADD.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ADDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ADDQ.L	#imm,<ea>	1(0/0)	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ADDX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
AND.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
AND.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ANDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ASL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
ASR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
BCHG	Dy,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)	—
BCHG	#imm,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)	—
BCLR	#imm,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)	—
BSET	#imm,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—	—
BTST	Dy,<ea>	2(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	—
BTST	#imm,<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	—	—	—
CMP.B	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
CMP.W	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
CMP.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
CMPI.B	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.W	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
DIVS.W	<ea>,Dx	20(0/0)	20(1/0)	20(1/0)	20(1/0)	20(1/0)	21(1/0)	20(1/0)	20(0/0)
DIVU.W	<ea>,Dx	20(0/0)	20(1/0)	20(1/0)	20(1/0)	20(1/0)	21(1/0)	20(1/0)	20(0/0)
DIVS.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
DIVU.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
EOR.L	Dy,<ea>	1(0/0)	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
EORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
LEA	<ea>,Ax	—	1(0/0)	—	—	1(0/0)	2(0/0)	1(0/0)	—
LSL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
LSR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)

Table 3-16. Two Operand Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
MOVEQ.L	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
OR.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
OR.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
REMS.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
REMU.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
SUB.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
SUB.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
SUBI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
SUBQ.L	#imm,<ea>	1(0/0)	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
SUBX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

3.3.5.5 Miscellaneous Instruction Execution Times

Table 3-17. Miscellaneous Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
CPUSHL	(Ax)	—	9(0/1)	—	—	—	—	—	—
CPUSHL	bc,Ax	—	18(0/1)	—	—	—	—	—	—
CPUSHL	dc,Ax	—	12(0/1)	—	—	—	—	—	—
CPUSHL	ic,Ax	—	18(0/1)	—	—	—	—	—	—
INTOUCH	(Ay)	—	19(1/0)	—	—	—	—	—	—
LINK.W	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
MOV3Q.L	#imm,<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
MOVE.L	Ay,USP	3(0/0)	—	—	—	—	—	—	—
MOVE.L	USP,Ax	3(0/0)	—	—	—	—	—	—	—
MOVE.W	CCR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,CCR	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.W	SR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,SR	4(0/0)	—	—	—	—	—	—	4(0/0) [2]
MOVEC	Ry,Rc	20(0/1)	—	—	—	—	—	—	—
MOVEM.L	<ea>, and list	—	n(n/0)	—	—	n(n/0)	—	—	—
MOVEM.L	and list,<ea>	—	n(0/n)	—	—	n(0/n)	—	—	—
MVS	<ea>,Dx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)

Table 3-17. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
MVZ	<ea>,Dx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
NOP		6(0/0)	—	—	—	—	—	—	—
PEA	<ea>	—	1(0/1)	—	—	1(0/1) [4]	2(0/1) [5]	1(0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STOP	#imm	—	—	—	—	—	—	—	6(0/0) [3]
TRAP	#imm	—	—	—	—	—	—	—	18(1/2)
TPF		1(0/0)	—	—	—	—	—	—	—
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPF.L		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	1(1/0)	—	—	—	—	—	—	—
WDDATA	<ea>	—	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	—
WDEBUG	<ea>	—	3(2/0)	—	—	3(2/0)	—	—	—

[1] The n is the number of registers moved by the MOVEM opcode.

[2] If a MOVE.W #imm,SR instruction is executed and imm[13] equals 1, the execution time is 1(0/0).

[3] The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

[4] PEA execution times are the same for (d16,PC).

[5] PEA execution times are the same for (d8,PC,Xn*SF).

3.3.5.6 EMAC Instruction Execution Times

Table 3-18. EMAC Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MAC.L	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.L	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0)[1]	—	—	—
MAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.W	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0)[1]	—	—	—
MOVE.L	<ea>y, Raccx	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	Raccy, Raccx	1(0/0)	—	—	—	—	—	—	—
MOVE.L	<ea>y, MACSR	8(0/0)	—	—	—	—	—	—	8(0/0)
MOVE.L	<ea>y, Rmask	7(0/0)	—	—	—	—	—	—	7(0/0)
MOVE.L	<ea>y,Raccext01	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	<ea>y,Raccext23	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	Raccx, <ea>x	1(0/0)[2]	—	—	—	—	—	—	—

Table 3-18. EMAC Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MOVE.L	MACSR, <ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Rmask, <ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Raccext01,<ea.x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Raccext23,<ea>x	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MSAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0)[1]	—	—	—
MSAC.W	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0)[1]	—	—	—
MULS.L	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	—	—	—
MULS.W	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
MULU.L	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	—	—	—
MULU.W	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)

[1] Effective address of (d16,PC) not supported

[2] Storing an accumulator requires one additional processor clock cycle when saturation is enabled, or fractional rounding is performed (MACSR[7:4] equals 1---, -11-, --11)

NOTE

The execution times for moving the contents of the Racc, Raccext[01,23], MACSR, or Rmask into a destination location <ea>x shown in this table represent the best-case scenario when the store instruction is executed and there are no load or M{S}AC instructions in the EMAC execution pipeline. In general, these store operations require only a single cycle for execution, but if preceded immediately by a load, MAC, or MSAC instruction, the depth of the EMAC pipeline is exposed and the execution time is four cycles.

3.3.5.7 Branch Instruction Execution Times

Table 3-19. General Branch Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BRA		—	—	—	—	1(0/1)[1]	—	—	—
BSR		—	—	—	—	1(0/1)[2]	—	—	—

Table 3-19. General Branch Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
JMP	<ea>	—	5(0/0)	—	—	5(0/0)[1]	6(0/0)	1(0/0)[1]	—
JSR	<ea>	—	5(0/1)	—	—	5(0/1)	6(0/1)	1(0/1)[2]	—
RTE		—	—	15(2/0)	—	—	—	—	—
RTS		—	—	2(1/0)[3] 9(1/0)[3] 8(1/0)[3]	—	—	—	—	—

Table 3-20. Bcc Instruction Execution Times

Opcode	Branch Cache Correctly Predicts Taken	Prediction Table Correctly Predicts Taken	Predicted Correctly as Not Taken	Predicted Incorrectly
Bcc	0(0/0)	1(0/0)	1(0/0)	8(0/0)

The following notes apply to the branch execution times:

1. For BRA and JMP <ea> instructions, where <ea> is (d16,PC) or xxx.wl, the branch acceleration logic of the IFP calculates the target address and begins prefetching the new path. Because the IFP and OEP are decoupled by the FIFO instruction buffer, the execution time can vary from one to three cycles, depending on the decoupling amount.

   For all other <ea> values of the JMP instruction, the branch acceleration logic is not used, and the execution times are fixed.
2. For BSR and JSR xxx.wl opcodes, the same branch acceleration mechanism is used to initiate the fetch of the target instruction. Depending on the amount of decoupling between the IFP and OEP, the resulting execution times can vary from 1 to 3 cycles.

   For the remaining <ea> values for the JSR instruction, the branch acceleration logic is not used, and the execution times are fixed.
3. For the RTS opcode, the timing depends on the prediction results of the hardware return stack:
   a) If predicted correctly, 2(1/0).
   b) If mispredicted, 9(1/0).
   c) If not predicted, 8(1/0).

Chapter 4
Memory Management Unit (MMU)

4.1 Introduction

This chapter describes the ColdFire virtual memory management unit (MMU), which provides virtual-to-physical address translation and memory access control. The MMU consists of memory-mapped control, status, and fault registers that provide access to translation-lookaside buffers (TLBs). Software can control address translation and access attributes of a virtual address by configuring MMU control registers and loading TLBs. With software support, the MMU provides demand-paged, virtual addressing.

4.1.1 Block Diagram

Figure 4-1 shows the placement of the MMU/TLB hardware. It follows a traditional model closely coupled to the processor local-memory controllers.

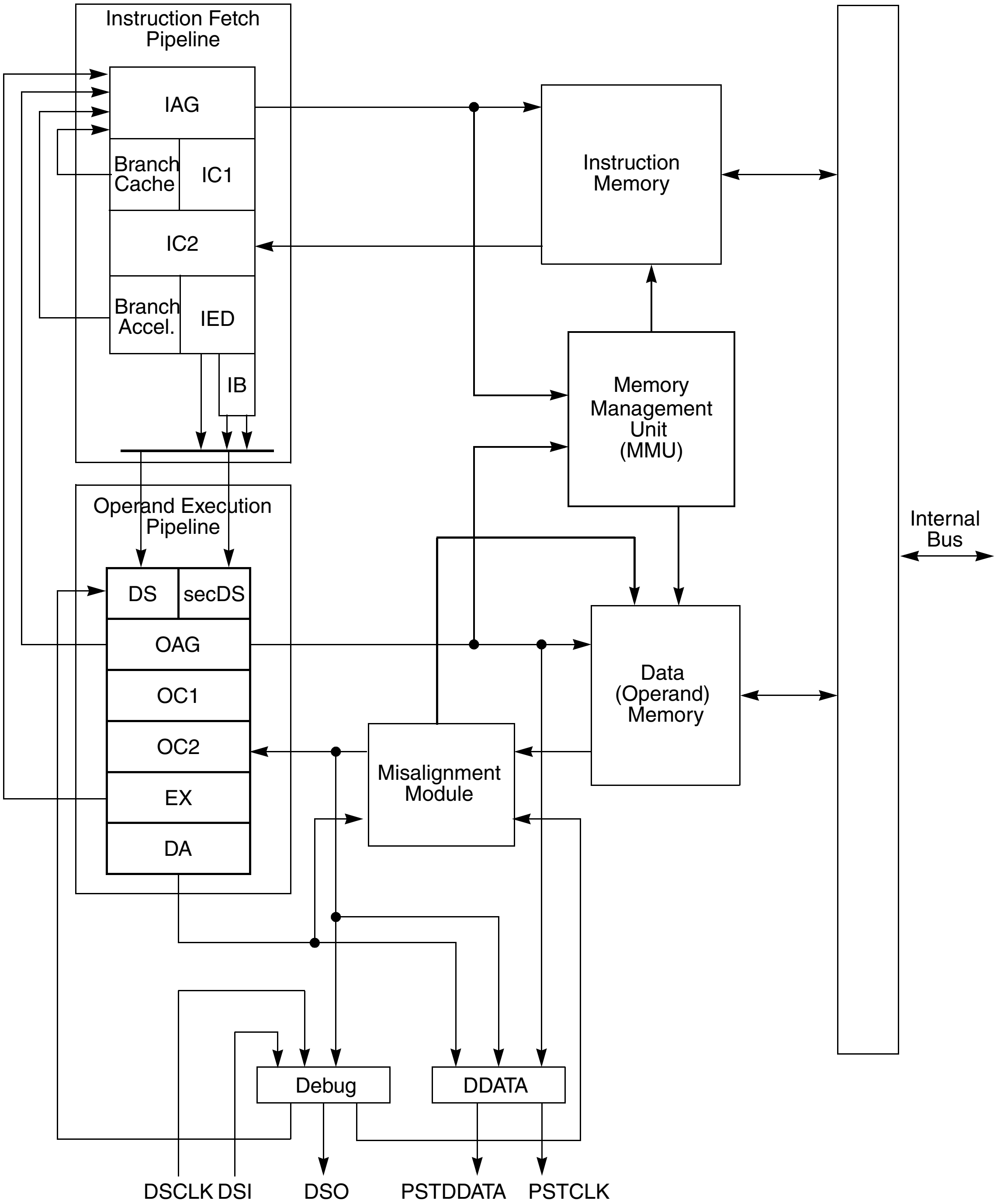


Figure 4-1. CF4 Processor Core Block with MMU

4.1.2 Features

The MMU has the following features:

- MMU memory-mapped control, status, and fault registers
  — Supports a flexible, software-defined virtual environment

  - Provides control and maintenance of TLBs
  - Provides fault status and recovery information functions
- Separate, 32-entry, fully associative instruction and data TLBs (Harvard TLBs)
  - Resides in the processor local bus-controller
  - Operates in parallel with internal memory
  - Suffers no performance penalty on TLB hits
  - Supports 4- and 8-Kbyte, and 1- and 16-Mbyte page sizes concurrently
  - Contains register-based TLB entries
- Core extensions:
  - User stack pointer
  - All access error exceptions are precise and recoverable
- Harvard TLB provides 97% of baseline performance on large embedded applications without MMU support

4.2 Memory Map/Register Definition

Access to the MMU memory-mapped region is controlled by MMUBAR, a 32-bit supervisor control register at 0x008 accessed using MOVEC or the serial BDM debug port. The *ColdFire Programmers Reference Manual* describes the MOVEC instruction.

MMUBAR holds the base address for the 64-Kbyte MMU memory map (Table 4-1). The MMU memory map area is not visible unless the MMUBAR is valid and must be referenced aligned. A large map portion is reserved for future use.

Table 4-1. MMU Memory Map

Address	Register	Width	Access	Reset Value	Section/Page
Rc[11:0] = 0x003[1]	ASID—Address Space ID	8	R/W	0x00	4.2.1/4-4
Rc[11:0] = 0x008[1]	MMUBAR—MMU Base Address Register	32	R/W	0x0000_0000	4.2.2/4-4
MMUBAR + 0x0000	MMUCR—MMU control register	32	R/W	0x0000_0000	4.2.3/4-5
MMUBAR + 0x0004	MMUOR—MMU operation register	32	R/W	0x0000_0000	4.2.4/4-6
MMUBAR + 0x0008	MMUSR—MMU status register	32	R/W	0x0000_0000	4.2.5/4-7
MMUBAR + 0x0010	MMUAR—MMU fault, test, or TLB address register	32	R/W	0x0000_0000	4.2.6/4-8
MMUBAR + 0x0014	MMUTR—MMU read/write TLB tag register	32	R/W	0x0000_0000	4.2.7/4-8

Table 4-1. MMU Memory Map (continued)

Address	Register	Width	Access	Reset Value	Section/Page
MMUBAR + 0x0018	MMUDR—MMU read/write TLB data register	32	R/W	0x0000_0000	4.2.8/4-9

[1] The address listed here represents the value of the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 34, “Debug Module.”

4.2.1 Address Space ID (ASID)

The address space ID (ASID) is located in a CPU space-control register. The 8-bit ASID value is mapped into CPU space at address 0x003 and is accessed using a MOVEC instruction. The *ColdFire Family Programmer’s Reference Manual* describes MOVEC.

Rc[11:0]: 0x003 (ASID) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R W	ID							
Reset:	0	0	0	0	0	0	0	0

Figure 4-2. Address Space ID (ASID)

Table 4-2. ASID Field Descriptions

Field	Description
7–0 ID	This 8-bit field is the current user ASID. The ASID is an extension to the virtual address. Address space 0x00 may be reserved for supervisor mode. See address space mode functionality in Section 4.2.3, “MMU Control Register (MMUCR).” The other 255 address spaces are used to tag user processes. The TLB entry ASID values are compared to this value for user mode unless the TLB entry is marked shared (MMUTR[SG] is set). The TLB entry ASID value may be compared to 0x00 for supervisor accesses.

4.2.2 MMU Base Address Register (MMUBAR)

The default reset state is an invalid MMUBAR; The MMU is disabled and the memory-mapped space is not visible.

Rc[11:0] 0x008 (MMUBAR) Access: Supervisor 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
R	BA																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	V
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-3. MMU Base Address Register (MMUBAR)

Table 4-3. MMUBAR Field Descriptions

Field	Description
31–16 BA	Base address. Defines the base address for the 64-Kbyte address space mapped to the MMU.
15–1	Reserved, must be cleared.
0 V	Valid. Indicates when MMUMBAR contents are valid. BA is not used unless V is set. 0 MMUBAR contents are not valid. 1 MMUBAR contents are valid.

4.2.3 MMU Control Register (MMUCR)

MMUCR contains the address space mode and virtual mode enable bits. The user must force pipeline synchronization after writing to this register. Therefore, all writes to this register must be immediately followed by a NOP instruction.

MMUBAR Offset: 0x000 (MMUCR)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ASM	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-4. MMU Control Register (MMUCR)

Table 4-4. MMUCR Field Descriptions

Bits	Description
31–2	Reserved, must be cleared.
1 ASM	Address space mode. Controls how the address space ID is used for TLB hits. 0 TLB entry ASID values are compared to the ASID register value for user or supervisor mode unless the TLB entry is marked shared (MMUTR[SG] = 1). The address space ID register value is the effective address space for all requests, supervisor and user. 1 Address space 0x00 is reserved for supervisor mode, and the effective address space is forced to 0x00 for all supervisor accesses. The other 255 address spaces are used to tag user processes. The TLB entry ASID values are compared to the ASID register for user mode unless the TLB entry is marked shared (SG = 1). The TLB entry ASID value is always compared to 0x00 for supervisor accesses. This allows two levels of sharing. All users, but not the supervisor, share an entry if SG is set and ASID does not equal 0. All users and the supervisor share an entry if SG is set and ASID equals 0
0 EN	Virtual mode enable. 0 Virtual mode is disabled 1 Virtual mode is enabled

4.2.4 MMU Operation Register (MMUOR)

MMUBAR Offset: 0x004 (MMUOR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	AA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	ITLB	ADR	R/W	0	0
W								STLB	CA	CNL	CAS				ACC	UAA
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-5. MMU Operation Register (MMUOR)

Table 4-5. MMUOR Field Descriptions

Field	Description
31–16 AA	TLB allocation address. This read-only field is maintained by MMU hardware. Its range and format depend on the TLB implementation (specific TLB size in entries, associativity, and organization). The access TLB function can use AA to read or write the addressed TLB entry. The MMU loads AA on the following three events: • On DTLB access errors, it loads the TLB entry address that caused the error. • If MMUOR[UAA] is set, it loads the address of the TLB entry chosen by the MMU for replacement. • If MMUOR[STLB] is set, it uses the data in MMUAR to search the TLB. If the TLB hits, it loads the address of the TLB entry that hits; if the TLB misses, it loads the TLB entry chosen by the MMU for replacement. The MMU never picks a locked entry for replacement, and TLB hits of locked entries do not update hardware replacement algorithm information. This is so access error handlers mapped with locked TLB entries do not influence the replacement algorithm. Further, TLB search operations do not update the hardware replacement algorithm information; TLB writes (loads) do update the hardware replacement algorithm information. The algorithm that chooses the allocation address depends on the TLB implementation (such as LRU, round-robin, pseudo-random).
15–9	Reserved, must be cleared.
8 STLB	Search TLB. STLB always reads as zero. 0 No operation 1 The MMU searches the TLB using data in MMUAR. This operation updates the probe TLB hit bit in the status register plus loads the AA field as described above.
7 CA	Clear all TLB entries. CA always reads as zero. 0 No operation 1 Clear all TLB entries and all hardware TLB replacement algorithm information.
6 CNL	Clear all non-locked TLB entries. Setting CNL clears all TLB entries that do not have locked bits. CNL always reads as zero. 0 No operation 1 Clear all non-locked TLB entries
5 CAS	Clear all non-locked TLB entries that match ASID. CAS always reads as a zero. 0 No operation 1 Clear all non-locked TLB entries that match ASID register
4 ITLB	ITLB operation. Used by TLB search and access operations that use the TLB allocation address. 0 MMU uses DTLB to search or update allocation address 1 MMU uses ITLB for of the allocation address searches and updates

Table 4-5. MMUOR Field Descriptions (continued)

Field	Description
3 ADR	TLB address select. Indicates which address to use when accessing the TLB. 0 Use the TLB allocation address for the TLB address 1 Use MMUAR for the TLB address
2 R/W	TLB access read/write select. Indicates whether to perform a read or a write when accessing the TLB. 0 Write 1 Read
1 ACC	MMU TLB access. This bit always reads as a zero. STLB is used for search operations. 0 No operation. ACC must be a zero to search the TLB. 1 The MMU reads or writes the TLB depending on R/W. For TLB reads, TLB tag and data results are loaded into MMUTR and MMUDR. For TLB writes, the contents of these registers are written to the TLB. The TLB is accessed using the TLB allocation address if ADR is zero or using MMUAR if ADR is set.
0 UAA	Update allocation address. UAA always reads as a zero. 0 No operation 1 MMU updates the allocation address field with the MMU's choice for the allocation address in the ITLB or DTLB depending on the ITLB instruction operation bit.

4.2.5 MMU Status Register (MMUSR)

MMUSR is updated on all data access faults and search TLB operations.

MMUBAR Offset: 0x008 (MMUSR)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SPF	RF	WF	0	HIT	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-6. MMU Status Register (MMUSR)

Table 4-6. MMUSR Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5 SPF	Supervisor-protect fault. Indicates if last data fault was a user-mode access that hit in a TLB entry with its supervisor protect bit set. 0 Last data access fault did not have a supervisor protect fault 1 Last data access fault had a supervisor protect fault
4 RF	Read-access fault. Indicates if last data fault was a data-read access that hit in a TLB entry without its read bit set. 0 Last data access fault did not have a read protect fault 1 Last data access fault had a read protect fault
3 WF	Write-access fault. Indicates if the last data fault was a data-write access that hit in a TLB entry without its write bit set. 0 Last data access fault did not have a write protect fault 1 Last data access fault had a write protect fault
2	Reserved, must be cleared.

Table 4-6. MMUSR Field Descriptions (continued)

Field	Description
1 HIT	Search TLB hit. Indicates if last data fault or last search TLB operation hit in the TLB. 0 Last data access fault or search TLB operation did not hit in the TLB 1 Last data access fault or search TLB operation hit in the TLB
0	Reserved, must be cleared.

4.2.6 MMU Fault, Test, or TLB Address Register (MMUAR)

The MMUAR format depends on how register is used.

MMUBAR Offset: 0x010 (MMUAR)

Access: User 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
R	FA																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-7. MMU Fault, Test, or TLB Address Register (MMUAR)

Table 4-7. MMUAR Field Descriptions

Field	Description
31–0 FA	Form address. • Written by the MMU with the virtual-address on DTLB misses and access faults. For this case, all 32 bits are address bits. • This register may be written with a virtual-address and address-attribute information for searching the TLB (MMUCR[STLB]). For this case, FA[31–1] are the virtual page number and FA[0] is the supervisor bit. The current ASID is used for the TLB search. • MMUAR can also be written with a TLB address for use with the access TLB function (using MMUCR[ACC]).

4.2.7 MMU Read/Write Tag Entry Registers (MMUTR)

Each TLB entry consists of a 32-bit TLB tag entry and a 32-bit TLB data entry. TLB entries are referenced through MMUTR and MMUDR registers. For read TLB accesses, the contents of the TLB tag and data entries referenced by the allocation address or MMUAR are loaded in MMUTR and MMUDR. TLB write accesses place MMUTR and MMUDR contents into the TLB tag and data entries defined by the allocation address or MMUAR.

The MMUTR register contains the virtual address tag, the address space ID (ASID), a shared page indicator, and the valid bit.

MMUBAR Offset: 0x014 (MMUTR)

Access: User 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
R W	VA																						ID								SG	V
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-8. MMU Read/Write TLB Tag Register (MMUTR)

Table 4-8. MMUTR Field Descriptions

Field	Description
31–10 VA	Virtual address. Defines the virtual address mapped by this entry. The number of bits used in TLB hit determination depends on the page-size field in the corresponding TLB data entry.
9–2 ID	Address space ID (ASID). This extension to the virtual address marks this entry as part of 1 of 256 possible address spaces. Address space 0x00 can be reserved for supervisor mode. The other 255 address spaces are used to tag user processes. TLB entry ASID values are compared to the ASID register value for user mode unless the TLB entry is marked shared (SG = 1). The TLB entry ASID value may be compared to 0x00 for supervisor accesses or to the ASID. The description of MMUCR[ASM] in Table 4-4 gives details on supervisor mode and ASID compares.
1 SG	Shared global. Indicates when the entry is shared among user address spaces. If an entry is shared, its ASID is not part of the TLB hit determination for user accesses. 0 This entry is not shared globally. 1 This entry is shared globally. **Note:** The ASID can determine supervisor mode hits to allow two sharing levels. If SG and MMUCR[ASM] are set and the ASID is not zero, all users (but not the supervisor) share an entry. If SG and MMUCR[ASM] are set and the ASID is zero, all users and the supervisor share an entry. The ASM description in Table 4-4 details supervisor mode and ASID compares.
0 V	Valid. Indicates when the entry is valid. Only valid entries generate a TLB hit. 0 Entry is not valid. 1 Entry is valid.

4.2.8 MMU Read/Write Data Entry Register (MMUDR)

The MMUDR register contains the physical address, page size, cache-mode field, supervisor-protect bit, read, write, execute permission bits, and lock-entry bit.

MMUBAR Offset: 0x018 (MMUTR)

Access: User 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
R	PA																						SZ		CM		SP	R	W	X	LK	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4-9. MMU Read/Write TLB Data Register (MMUDR)

Table 4-9. MMUDR Field Descriptions

Field	Descriptions
31–10 PA	Physical address. Defines the physical address mapped by this entry. The number of bits used to build the effective physical address if this TLB entry hits depends on the page size field.
9–8 SZ	Page size. Page size for this entry: 00 1 Mbyte: VA[31–20] used for TLB hit 01 4 Kbytes: VA[31–12] used for TLB hit 10 8 Kbytes: VA[31–13] used for TLB hit 11 16 Mbytes: VA[31–24] used for TLB hit
7–6 CM	Cache mode. **Instruction cache modes:** 1*x* Page is non-cacheable. 0*x* Page is cacheable. **Data cache modes:** 00 Page is cacheable write-through. 01 Page is cacheable copy-back. 10 Page is non-cacheable precise. 11 Page is non-cacheable imprecise.
5 SP	Supervisor protect. Controls user mode access to the page mapped by this entry. 0 Entry is not supervisor protected. 1 Entry is supervisor protected. An attempted user mode access that matches this entry generates an access error exception.
4 R	Read access enable. Indicates if data read accesses to this entry are allowed. If a Harvard TLB implementation is used, this bit is a don’t care for the ITLB. This bit is ignored on writes and always reads as zero for the ITLB. 0 Do not allow data read accesses. Attempted data read accesses that match this entry generate an access error exception. 1 Allow data-read accesses.
3 W	Write access enable. Indicates if data write accesses are allowed to this entry. If separate ITLB and DTLBs are used, this bit is a don’t care for the ITLB. This bit is ignored on writes and always reads as zero for the ITLB. 0 Do not allow data write accesses. Attempted data write accesses that match this entry generate an access error exception. 1 Allow data-write accesses.
2 X	Execute access enable. Indicates if instruction fetches to this entry are allowed. If separate ITLB and DTLBs are used, this bit is a don’t care for the DTLB. This bit is ignored on writes and reads as zero for the DTLB. 0 Do not allow instruction fetches. Attempted instruction fetches that match this entry cause an access error exception. 1 Allow instruction-fetch accesses.
1 LK	Lock entry bit. Indicates if this entry is included in the replacement algorithm. TLB hits of locked entries do not update replacement algorithm information. 0 Include this entry when determining the best entry for a TLB allocation. 1 Do not allow this entry to be selected by the replacement algorithm.
0	Reserved, must be cleared.

4.3 Functional Description

The ColdFire MMU provides a virtual address, demand-paged memory architecture. The MMU supports hardware address translation acceleration using software-managed TLBs. It enforces permission checking on a per-memory request basis, and has control, status, and fault registers for MMU operation.

4.3.1 Virtual Memory Management Architecture

The ColdFire memory management architecture provides a demand-paged, virtual-address environment with hardware address translation acceleration. It supports supervisor/user, read, write, and execute permission checking on a per-memory request basis.

The architecture defines the MMU TLB, associated control logic, TLB hit/miss logic, address translation based on the TLB contents, and access faults due to TLB misses and access violations. It intentionally leaves some virtual environment details undefined to maximize the software-defined flexibility. These include the exact structure of the memory-resident pointer descriptor/page descriptor tables, the base registers for these tables, the exact information stored in the tables, the methodology (if any) for access maintenance, and written information on a per-page basis.

4.3.1.1 MMU Architecture Features

To add optional virtual-addressing support, demand-page support, permission checking, and hardware address translation acceleration to the ColdFire architecture, the MMU architecture features:

- Addresses from the core to the MMU are treated as physical or virtual addresses.
- The address access control logic, address attribute logic, internal memories, and internal to external memory bus controller function as in previous ColdFire versions with the addition of MMU.
- MMU, its TLB, and associated control reside in the processor local bus logic.
- MMU appears as a memory-mapped device in the processor local bus space. Information for access error fault processing is stored in MMU.
- A precise processor local-bus fault (transfer-error acknowledge) signals the core on translation (TLB miss) and access faults. The core supports an instruction restart model for this fault class. This structure uses the existing ColdFire access error fault vector and needs no new ColdFire exception stack frames.
- New ACR bits improve address granularity, supervisor mode protection, and memory functionality for physical and virtual address environments.

4.3.1.2 MMU Architecture Implementation

This section describes ColdFire design additions and changes for the MMU architecture. It includes precise faults, MMU access, virtual mode, virtual memory references, instruction and data cache addresses, supervisor/user stack pointers, access error stack frame additions, expanded control register space, ACR address improvements, supervisor protection, and debugging in a virtual environment.

4.3.1.2.1 Precise Faults

The MMU architecture performs virtual-to-physical address translation and permission checking in the core. To support demand-paging, the core design provides a precise, recoverable fault for all processor local bus references.

4.3.1.2.2 MMU Access

The MMU TLB control registers are memory-mapped. The TLB entries are read and written indirectly through MMU control registers. Memory space for these resources is defined by a new supervisor program model register, the MMU base-address register (MMUBAR). This defines a supervisor-mode, data-only space. It has the highest priority for the data-processor local-bus address-mode determination.

4.3.1.2.3 Virtual Mode

Every processor local-bus instruction and data reference is a virtual or physical address mode access. The following are always physical accesses:

- All addresses for special mode (interrupt acknowledges, emulator mode operations, etc.) accesses
- All addresses if the MMU is not enabled

If the MMU is enabled, the address mode for normal accesses is determined by the MMUBAR, RAMBARs, and ACRs in the priority order listed:

- Addresses that hit in these registers are treated as physical references. These addresses are not translated and their address attributes are sourced from the highest priority mapping register they hit.
- If an address hits none of these mapping registers, it is a virtual address and is sent to the MMU. If the MMU is enabled, the default CACR information is not used.

4.3.1.2.4 Virtual Memory References

The ColdFire MMU architecture references the MMU for all virtual mode accesses to the processor local bus. MMU, SRAM and ACR memory spaces are treated as physical address spaces and all permissions applying to these spaces are contained in the respective mapping register. The virtual mode access either hits or misses in the TLB of the MMU. A TLB miss generates an access fault in the processor, allowing software to either load the appropriate translation into the TLB and restart the faulting instruction or abort the process. Each TLB hit checks permissions based on the access control information in the referenced TLB entry.

4.3.1.2.5 Instruction and Data Cache Addresses

For a given page size, virtual address bits that reference within a page are called the in-page address. All bits above this are the virtual page number. Likewise, the physical address has a physical page number and in-page address bits. Virtual and physical in-page address bits are the same; the MMU translates the virtual page number to the physical page number.

Instruction and data caches are accessed with the untranslated processor local-bus address. The translated address is used for cache allocation. That is, caches are virtual-address accessed and physical-address tagged. If instruction and data cache addresses are not larger than the in-page address for the smallest active MMU page, the cache is physically accessed; if they are larger, the cache can have aliasing problems between virtual and cache addresses. Software handles these problems by forcing the virtual address to be equal to the physical address for those bits addressing the cache, but above the in-page address of the smallest active page size. The number of these bits depends on cache and page sizes.

Caches are addressed with the virtual address (the cache uses synchronous memory elements), and an access starts at the rising-clock edge of the first processor local bus pipeline stage. The MMU provides a physical address midway through this cycle.

If the cache-set address has fewer bits than the in-page address, the cache is considered physically addressed because these bits are the same in the virtual and physical addresses. If the cache set address has more bits than the in-page address, one or more of the low-order virtual page number bits are used to address the cache. The MMU translates these bits; the resulting low-order physical page number bits are used to determine cache hits.

Address aliasing problems occur when two virtual addresses access one physical page. This is generally allowed and, if the page is cacheable, one coherent copy of the page image is mapped in the cache at any time.

If multiple virtual addresses pointing to the same physical address differ only in the low-order virtual page number bits, conflicting copies can be allocated. For an 8-Kbyte, 4-way, set-associative cache with a 16-byte line size, the cache set address uses address bits 10–4. If virtual addresses 0x0_1000 and 0x0_1400 are mapped to physical address 0x0_1000, using virtual address 0x0_1000 loads cache set 0x00; using virtual address 0x0_1400 loads cache set 0x40. This puts two copies of the same physical address in the cache, making this memory space not coherent. To avoid this problem, software must force low-order virtual page number bits to be equal to low-order physical address bits for all bits used to address the cache set.

4.3.1.2.6 Supervisor/User Stack Pointers

To isolate supervisor and user modes, the Version 4 ColdFire core implements two A7 register stack pointers: one for supervisor mode (SSP) and one for user mode (USP). Two former M68000 family-privileged instructions to load and store the user stack pointer are restored in the instruction set architecture.

4.3.1.2.7 Access Error Stack Frame

Processor local bus accesses that fault (that is, terminate with a transfer error acknowledge) to generate an access error exception. MMU TLB misses and access violations use the same fault. New fault status field (FS) encodings in the exception stack frame signal TLB misses on the following to quickly determine if a fault was due to a TLB miss or another type of access error:

- Instruction fetch
- Instruction extension fetch
- Data read
- Data write

See Section 4.3.3.3, “Access Error Stack Frame Additions,” for more information.

4.3.1.2.8 Expanded Control Register Space

The MMU base-address register (MMUBAR) is added for ColdFire virtual mode. Like other control registers, it can be accessed from the debug module or written using the privileged MOVEC instruction. See Section 4.2.2, “MMU Base Address Register (MMUBAR).”

4.3.1.2.9 Changes to ACRs and CACR

New ACR and CACR bits improve address granularity and supervisor mode protection and memory functionality for physical- and virtual- address environments.

Table 4-10. New ACR and CACR Bits

Field	Description
ACR*n*[10] AMM	Address mask mode. Determines access to the associated address space. 0 The ACR hit function is the same as previous versions, allowing control of a 16-Mbyte or greater memory region. 1 The upper 8 bits of the address and ACR are compared without a mask function; bits 23–20 of the address and ACR are compared masked by ACR[19–16], allowing control of a 1- to 16-Mbyte region. Reset value is 0.
ACR*n*[3] SP	Supervisor protect. Determines access to the associated address space. 0 Supervisor and user access allowed. 1 Only supervisor access allowed. Attempted user access causes an access error exception. Reset value is 0.
CACR[23] DDSP	Default data supervisor protect. Determines access to the associated data space. 0 Supervisor and user access allowed. 1 Only supervisor access allowed. Attempted user access causes an access error exception. Reset value is 0.
CACR[7] DISP	Default instruction supervisor protect. Determines access to the associated instruction space. 0 Supervisor and user access allowed. 1 Only supervisor access allowed. Attempted user access causes access error exception Reset value is 0.

4.3.1.2.10 ACR Address Improvements

The ACR registers provide a 16-Mbyte address window. For a given request address, if the ACR is valid and the request mode matches the mode specified in the supervisor mode field (ACR*n*[S]), hit determination is specified as:

```
ACRx_Hit = 0;
if ((address[31:24] and  ~ACRn[23:16]) == (ACRn[31:24] and  ~ACRn[23:16]))
         ACRx_Hit = 1;
```

With this hit function, ACRs can assign address attributes for user or supervisor requests to memory spaces of at least 16 Mbytes (through the address mask). With the MMU definition, the ACR hit function is improved by the address mask mode bit (ACR*n*[AMM]), which supports finer address granularity. See Table 4-10.

The revised hit determination becomes:

```
ACRx_Hit = 0;
if (ACRn[10] == 1)
         if ((address[31-24] == ACRn[31-24])) &&
                   ((address[23-20] and  ~ACRn[19-16]) == (ACRn[23-20] and  ~ACRn[19-16])))
                            ACRx_Hit = 1;
else if (address[31-24] and  ~ACRn[23-16]) == (ACRn[31-24] and  ~ACRn[23-16]))
                   ACRx_Hit = 1;
```

4.3.1.2.11 Supervisor Protection

Each instruction or data reference is a supervisor or user access. The CPU's status register supervisor bit (SR[S]) determines the operating mode. New ACR and CACR bits protect supervisor space. See Table 4-10.

4.3.2 Debugging in a Virtual Environment

To support debugging in a virtual environment, numerous enhancements are implemented in the ColdFire debug architecture. These enhancements are collectively called debug revision D and primarily relate to the addition of an 8-bit address space identifier (ASID) to yield a 40-bit virtual address. This expansion affects two major debug functions:

- The ASID is optionally included in the hardware breakpoint registers specification. For example, the four PC breakpoint registers are expanded by 8 bits each, so that a specific ASID value can be part of the breakpoint instruction address. Likewise, data address/data breakpoint registers are expanded to include an ASID value. The new control registers define if and how the ASID is included in the breakpoint comparison trigger logic.
- The debug module implements the concept of ownership trace in which an ASID value can be optionally displayed as part of real-time trace. When enabled, real-time trace displays instruction addresses on any change-of-flow instruction that is not absolute or PC-relative. For debug revision D architecture, the address display is expanded to include ASID contents optionally, thus providing the complete instruction virtual address on these instructions. Additionally, when a SYNC_PC serial BDM command is loaded from the external development system, the processor displays the complete virtual-instruction address, including the 8-bit ASID value.

The MMU control registers are accessible through serial BDM commands. See Chapter 34, "Debug Module."

4.3.3 Virtual Memory Architecture Processor Support

To support the MMU, enhancements have been made to the exception model, the stack pointers, and the access error stack frame.

4.3.3.1 Precise Faults

To support demand-paging, all memory references require precise, recoverable faults. The ColdFire instruction-restart mechanism ensures a faulted instruction restarts from the beginning of execution; in other words, no internal state information is saved when an exception occurs and none is restored when the handler ends. Given the PC address defined in the exception stack frame, the processor reestablishes program execution by transferring control to the given location as part of the RTE (return from exception) instruction.

For a detailed description, see Section 3.3.4.16, "Precise Faults."

4.3.3.2 Supervisor/User Stack Pointers

To provide the required isolation among these operating modes as dictated by a virtual memory management scheme, a user stack pointer (A7–USP) is added. The appropriate stack pointer register (SSP, USP) is accessed as a function of the processor's operating mode.

In addition, the following two privileged M68000 family instructions to load/store the USP are added to the ColdFire instruction set architecture:

```
move.l   Ay,USP    # move to USP: opcode = 0x4E6{0-7}
move.l   USP,Ax    # move from USP: opcode = 0x4E6{8-F}
```

The address register number is encoded in the three low-order bits of the opcode.

These instructions are described in detail in Section 4.3.9, "MMU Instructions."

4.3.3.3 Access Error Stack Frame Additions

ColdFire exceptions generate a standard 2-longword stack frame, signaling the contents of the SR and PC at the time of the exception, the exception type, and a 4-bit fault status field (FS). The first longword contains the 16-bit format/vector word (F/V) and the 16-bit status register. The second contains the 32-bit program counter address of the faulted instruction. For more information, see Section 3.3.3.1, "Exception Stack Frame Definition."

	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
SSP →	Format	FS[3:2]	Vector	FS[1:0]	Status Register
+ 0x4	Program Counter				

Figure 4-10. Exception Stack Frame Form

The FS field is used for access and address errors. To optimize TLB miss-exception handling, new FS encodings (as shown in Table 4-11) allow quick error classification.

Table 4-11. Fault Status Encodings

FS[3:0]	Definition
0000	Not an access or address error
0001 – 0011	Reserved
0100	Error (for example, protection fault) on instruction fetch
0101	TLB miss on opword of instruction fetch (New for MMU)
0110	TLB miss on extension word of instruction fetch (New for MMU)
0111	IFP access error while executing in emulator mode (New for MMU)
1000	Error on data write
1001	Attempted write of protected space
1010	TLB miss on data write (New for MMU)
1011	Reserved
1100	Error on data read

Table 4-11. Fault Status Encodings (continued)

FS[3:0]	Definition
1101	Attempted read, read-modify-write of protected space (New for MMU)
1110	TLB miss on data read, or read-modify-write (New for MMU)
1111	OEP access error while executing in emulator mode (New for MMU)

4.3.4 Effective Address Attribute Determination

The ColdFire core generates an effective memory address for all instruction fetches and data read and write memory accesses. The previous ColdFire memory access control model was based strictly on physical addresses. Every memory request address is a physical address analyzed by this memory access control logic and assigned address attributes, including:

- Cache mode
- SRAM enable information
- Write protect information
- Write mode information

These attributes control processing of the memory request. The address itself is not affected by memory access control logic.

Instruction and data references base effective address attributes and access mode on the instruction type and the effective address. There are two types of accesses:

- Special mode accesses, including interrupt acknowledges, reads/writes to program-visible control registers (CACR, RAMBARs, and ACRs), cache-control commands (CPUSHL and INTOUCH), and emulator-mode operations. These accesses have the following attributes:
  - Non-cacheable
  - Precise
  - No write protection

  Unless the CPU space/IACK mask bit is set, interrupt acknowledge cycles and emulator mode operations are allowed to hit in RAMBAR. All other operations are normal mode accesses.
- Normal mode accesses. For these accesses, an effective cache mode, precision, and write-protection are calculated for each request.

For data, a normal mode access address is compared with the following priority, from highest to lowest: RAMBAR, ACR0, and ACR1. If no match is found, default attributes in the CACR are used. The priority for instruction accesses is RAMBAR, ACR2, and ACR3. Again, if no match is found, default CACR attributes are used.

Only the test-and-set (TAS) instruction generates a normal mode access with implied cache mode and precision. TAS is a special, byte-sized, read-modify-write instruction used in synchronization routines. A TAS data access that does not hit in the RAMBAR is non-cacheable and precise. TAS uses the normal effective write protection.

If the MMU is enabled, it adds two factors for calculating effective address attributes:

- MMUBAR defines a memory-mapped, privileged data-only space with the highest priority in effective address attribute calculation for the data bus (that is, the MMUBAR has priority over RAMBAR).
- If virtual mode is enabled, any normal mode access that does not hit in the MMUBAR, RAMBAR or ACRs is considered a normal mode virtual address request and generates its access attributes from the MMU. For this case, the default CACR address attributes are not used.

The MMU also uses TLB contents to perform virtual-to-physical address translation.

4.3.5 MMU Functionality

The MMU provides virtual-to-physical address translation and memory access control. The MMU consists of memory-mapped, control, status, and fault registers, and a TLB that can be accessed through MMU registers. Supervisor software can access these resources through MMUBAR. Software can control address translation and virtual address access attributes by configuring MMU control registers and loading the MMU's TLB, which functions as a cache, associating virtual addresses to corresponding physical addresses and providing access attributes. Each TLB entry maps a virtual page. Several page sizes are supported. Features such as clear-all and probe-for-hit help maintain TLBs.

Fault-free, virtual address accesses that hit in the TLB incur no pipeline delay. Accesses that miss the TLB or hit the TLB but violate an access attribute generate an access-error exception. On an access error, software can reference address and information registers in the MMU to retrieve data. Depending on the fault source, software can obtain and load a new TLB entry, modify the attributes of an existing entry, or abort the faulting process.

4.3.6 MMU TLB

Each TLB entry consists of two 32-bit fields. The first is the TLB tag entry, and the second is the TLB data entry. TLB entries can be read and written through MMU registers. TLB contents are unaffected by reset.

4.3.7 MMU Operation

The processor sends instruction-fetch requests and data read/write requests to the MMU internal bus in the instruction- and operand-address generation cycles (IAG and OAG). The processor local bus controller and memories occupy the next two pipeline stages, instruction fetch cycles 1 and 2 (IC1 and IC2) and operand fetch cycles 1 and 2 (OC1 and OC2). For late writes, optional data pipeline stages are added to the processor local bus controller as well as any writable memories.

Table 4-12 shows the association between internal memory pipeline stages and the processor's pipeline structures (Figure 4-1).

Table 4-12. Version 4 Processor Local Bus Memory Pipelines

Processor Local Bus Memory Pipeline Stage	Instruction Fetch Pipeline	Operand Execution Pipeline
J stage	IAG	OAG
KC1 stage	IC1	OC1

Table 4-12. Version 4 Processor Local Bus Memory Pipelines (continued)

Processor Local Bus Memory Pipeline Stage	Instruction Fetch Pipeline	Operand Execution Pipeline
KC2 stage	IC2	OC2
Operand-execute stage	n/a	EX
Late-write stage	n/a	DA

Version 4 ColdFire processor local buses use the same 2-cycle read pipeline developed for Version 3 ColdFire cores. Each processor local bus has 32-bit address and 32-bit read data paths. Version 4 ColdFire cores use synchronous memory elements for all memory-control units. To support this, certain control information and all address bits are sent on the processor local buses at the end of the cycle before the initial bus access cycle (J cycle). The data processor local bus has an additional 32-bit write data path. For processor-store operations, Version 4 ColdFire uses a late-write strategy, which can require two additional data processor local bus cycles. This strategy yields the processor local bus pipeline behavior described in Table 4-13.

Table 4-13. Processor Local Bus Pipeline Cycles

Cycle	Description
J	Control and partial address broadcast (to start synchronous memories)
KC1	Complete address and control broadcast plus MMU information. During this cycle, all memory element read operations are performed; memory arrays are accessed.
KC2	Select appropriate memory as source, return data to processor, handle cache misses or hold processor local bus pipeline as needed.
EX	Optional write stage, pipeline address and control for store operations.
DA	Data available for stores from processor; memory element update occurs in the next cycle.

The processor core contains two independent memory unit access controllers and two independent processor local bus controllers. Each instruction and data processor local bus request is analyzed to see which, if any, memory controller is referenced. This information, along with cache mode, store precision, and fault information, is sourced during KC1.

The MMU is referenced concurrently with the memory unit access controllers. It has two independent control sections to process simultaneously an instruction and data processor local bus request. Figure 4-1 shows how the MMU and memory unit access controllers fit in the processor local bus pipeline. As the diagram shows, core address and attributes access the mapping registers and the MMU. By the middle of the KC1 cycle, the physical-memory address is available along with its corresponding access control.

Figure 4-11 shows more details of the MMU structure. At the beginning of the KC1 pipeline stage, the TLB is accessed so the resulting physical address can be sourced to the cache controllers to factor into the cache hit/miss determination. This is required because caches are virtually indexed but physically mapped.

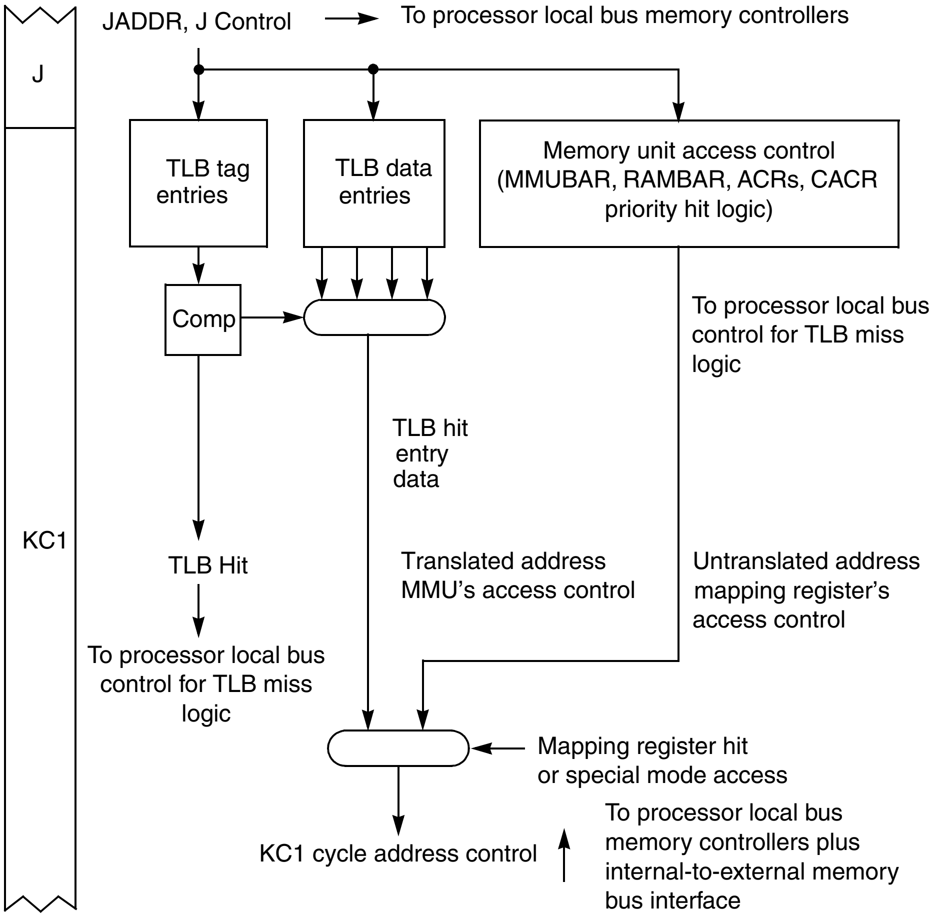

JADDR, J Control
To processor local bus memory controllers
J
KC1
TLB tag entries
TLB data entries
Memory unit access control (MMUBAR, RAMBAR, ACRs, CACR priority hit logic)
Comp
To processor local bus control for TLB miss logic
TLB hit entry data
TLB Hit
Translated address MMU's access control
Untranslated address mapping register's access control
To processor local bus control for TLB miss logic
Mapping register hit or special mode access
KC1 cycle address control
To processor local bus memory controllers plus internal-to-external memory bus interface

Figure 4-11. Processor Local Bus Address and Attributes Generation

4.3.8 MMU Implementation

The MMU implements a 64-entry full-associative Harvard TLB architecture with 32-entry ITLB and DTLB. This section details the operation and looks at the size, frequency, miss rate, and miss recovery time of this TLB implementation.

4.3.8.1 TLB Address Fields

Because the TLB has a total of 64 entries (32 each for the instruction and data TLBs), a 6-bit address field is necessary. TLB addresses 0–31 reference the ITLB; TLB addresses 32–63 reference the DTLB.

In the MMUOR register, bits 0–5 of the TLB allocation address (AA[5–0]) have this address format. The remaining TLB allocation address bits (AA[15–6]) are ignored on updates and always read as zero.

When the MMUAR register is used for a TLB address, bits FA[5–0] also have this address format. The remaining form address bits (FA[31–6]) are ignored when this register is used for a TLB address.

4.3.8.2 TLB Replacement Algorithm

The instruction and data TLBs provide low-latency access to recently used instruction and operand translation information. The ITLBs and DTLBs are 32-entry fully associative caches. The 32 ITLB entries are searched on each instruction reference; the 32 DTLB entries are searched on each operand reference.

The TLBs are software controlled. The TLB clear-all function clears valid bits on every TLB entry and resets the replacement logic. A new valid entry loaded in the TLBs may be designated as locked and unavailable for allocation. TLB hits to locked entries do not update replacement algorithm information.

When a new TLB entry needs to be allocated, the user can specify the exact TLB entry to be updated (through MMUOR[ADR] and MMUAR) or let TLB hardware pick the entry to update based on the replacement algorithm. A pseudo least recently used (PLRU) algorithm picks the entry to replace on a TLB miss. The algorithm works as follows:

- If any element is empty (non-valid), use the lowest empty element as the allocate entry (entry 0 before 1, 2, 3, and so on).
- If all entries are valid, use the entry indicated by the PLRU as the allocate entry.

The PLRU algorithm uses 31 most recently used state bits per TLB to track the TLB hit history. Table 4-14 lists these state bits.

Table 4-14. PLRU State Bits

State Bits	Meaning
rdRecent31To16	A 1 indicates 31To16 is more recent than 15To00
rdRecent31To24	A 1 indicates 31To24 is more recent than 23To16
rdRecent15To08	A 1 indicates 15To08 is more recent than 07To00
rdRecent31To28	A 1 indicates 31To28 is more recent than 27To24
rdRecent23To20	A 1 indicates 23To20 is more recent than 19To16
rdRecent15To12	A 1 indicates 15To12 is more recent than 11To08
rdRecent07To04	A 1 indicates 07To04 is more recent than 03To00
rdRecent31To30	A 1 indicates 31To30 is more recent than 29To28
rdRecent27To26	A 1 indicates 27To26 is more recent than 25To24
rdRecent23To22	A 1 indicates 23To22 is more recent than 21To20
rdRecent19To18	A 1 indicates 19To18 is more recent than 17To16
rdRecent15To14	A 1 indicates 15To14 is more recent than 13To12
rdRecent11To10	A 1 indicates 11To10 is more recent than 09To08
rdRecent07To06	A 1 indicates 07To06 is more recent than 05To04
rdRecent03To02	A 1 indicates 03To02 is more recent than 01To00
rdRecent31	A 1 indicates 31 is more recent than 30
rdRecent29	A 1 indicates 29 is more recent than 28
rdRecent27	A 1 indicates 27 is more recent than 26

Table 4-14. PLRU State Bits (continued)

State Bits	Meaning
rdRecent25	A 1 indicates 25 is more recent than 24
rdRecent23	A 1 indicates 23 is more recent than 22
rdRecent21	A 1 indicates 21 is more recent than 20
rdRecent19	A 1 indicates 19 is more recent than 18
rdRecent17	A 1 indicates 17 is more recent than 16
rdRecent15	A 1 indicates 15 is more recent than 14
rdRecent13	A 1 indicates 13 is more recent than 12
rdRecent11	A 1 indicates 11 is more recent than 10
rdRecent09	A 1 indicates 09 is more recent than 08
rdRecent07	A 1 indicates 07 is more recent than 06
rdRecent05	A 1 indicates 05 is more recent than 04
rdRecent03	A 1 indicates 03 is more recent than 02
rdRecent01	A 1 indicates 01 is more recent than 00

Binary state bits are updated on all TLB write (load) operations, as well as normal non-locked entries ITLB and DTLB hits. Also, if all entries in a binary state are locked, then that state is always set. That is, if entries 15, 14, 13, and 12 were locked, LRU state bit rdRecent15To14 is forced to 1.

For a completely valid TLB, binary state information determines the LRU entry. The replacement algorithm is deterministic and, for the case of a full TLB (with no locked entries and always touching new pages), the replacement entry repeats every 32 TLB loads.

4.3.8.3 TLB Locked Entries

Figure 4-12 is a ColdFire MMU Harvard TLB block diagram.

For TLB miss faults, the instruction restart model re-executes an instruction on returning from the exception handler. An instruction can touch two instruction pages (a 32- or 48-bit instruction can straddle two pages) or four data pages (a memory-to-memory word or longword move where misaligned source and destination operands straddle two pages). Therefore, one instruction may take two ITLB misses and allocate two ITLB pages before completion. Likewise, one instruction may require four DTLB misses and allocate four DTLB pages. Because of this, a pool of unlocked TLB entries must be available if virtual memory is used.

The above examples show the fewest entries needed to guarantee an instruction can complete execution. For good MMU performance, more unlocked TLB entries should be available.

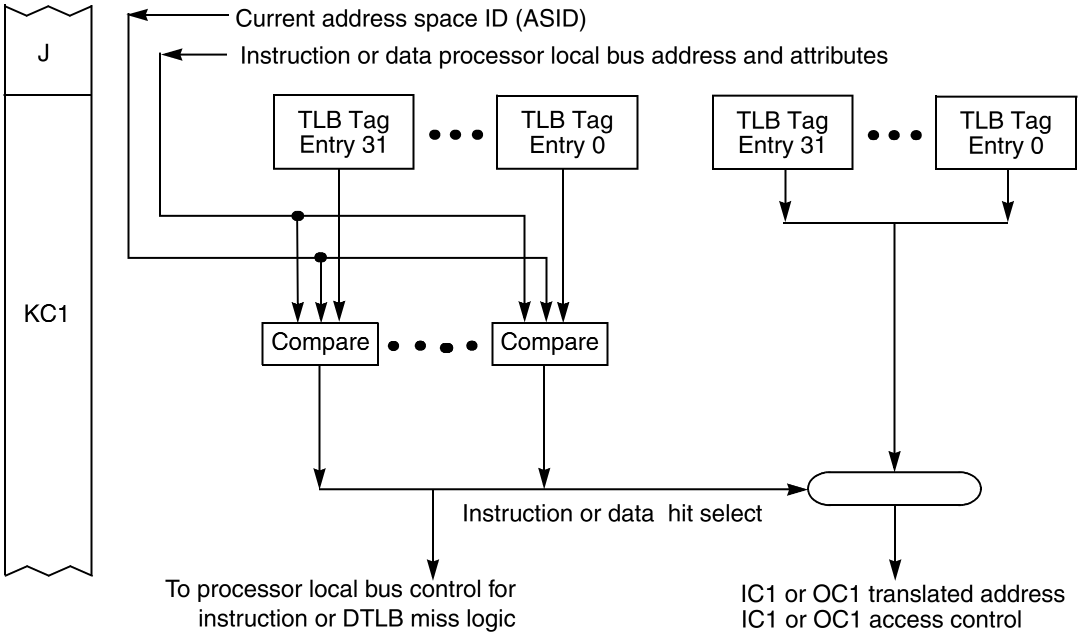

J
KC1
Current address space ID (ASID)
Instruction or data processor local bus address and attributes
TLB Tag Entry 31
TLB Tag Entry 0
TLB Tag Entry 31
TLB Tag Entry 0
Compare
Compare
Instruction or data hit select
To processor local bus control for instruction or DTLB miss logic
IC1 or OC1 translated address
IC1 or OC1 access control

Figure 4-12. Version 4 ColdFire MMU Harvard TLB

4.3.9 MMU Instructions

The MOVE to USP and MOVE from USP instructions are added for accessing the USP. Refer to the *ColdFire Programmer's Reference Manual* for more information.

Chapter 5
Enhanced Multiply-Accumulate Unit (EMAC)

5.1 Introduction

This chapter describes the functionality, microarchitecture, and performance of the enhanced multiply-accumulate (EMAC) unit in the ColdFire family of processors.

5.1.1 Overview

The EMAC design provides a set of DSP operations that can improve the performance of embedded code while supporting the integer multiply instructions of the baseline ColdFire architecture.

The MAC provides functionality in three related areas:

1. Signed and unsigned integer multiplication
2. Multiply-accumulate operations supporting signed and unsigned integer operands as well as signed, fixed-point, and fractional operands
3. Miscellaneous register operations

The ColdFire family supports two MAC implementations with different performance levels and capabilities. The original MAC features a three-stage execution pipeline optimized for 16-bit operands, with a 16x16 multiply array and a single 32-bit accumulator. The EMAC features a four-stage pipeline optimized for 32-bit operands, with a fully pipelined 32 × 32 multiply array and four 48-bit accumulators.

The first ColdFire MAC supported signed and unsigned integer operands and was optimized for 16x16 operations, such as those found in applications including servo control and image compression. As ColdFire-based systems proliferated, the desire for more precision on input operands increased. The result was an improved ColdFire MAC with user-programmable control to optionally enable use of fractional input operands.

EMAC improvements target three primary areas:

- Improved performance of 32 × 32 multiply operation.
- Addition of three more accumulators to minimize MAC pipeline stalls caused by exchanges between the accumulator and the pipeline’s general-purpose registers
- A 48-bit accumulation data path to allow a 40-bit product, plus 8 extension bits increase the dynamic number range when implementing signal processing algorithms

The three areas of functionality are addressed in detail in following sections. The logic required to support this functionality is contained in a MAC module (Figure 5-1).

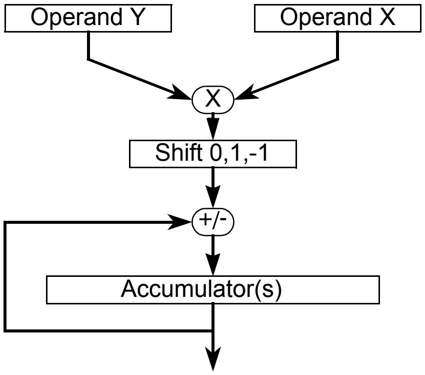


Figure 5-1. Multiply-Accumulate Functionality Diagram

5.1.1.1 Introduction to the MAC

The MAC is an extension of the basic multiplier in most microprocessors. It is typically implemented in hardware within an architecture and supports rapid execution of signal processing algorithms in fewer cycles than comparable non-MAC architectures. For example, small digital filters can tolerate some variance in an algorithm's execution time, but larger, more complicated algorithms such as orthogonal transforms may have more demanding speed requirements beyond scope of any processor architecture and may require full DSP implementation.

To balance speed, size, and functionality, the ColdFire MAC is optimized for a small set of operations that involve multiplication and cumulative additions. Specifically, the multiplier array is optimized for single-cycle pipelined operations with a possible accumulation after product generation. This functionality is common in many signal processing applications. The ColdFire core architecture is also modified to allow an operand to be fetched in parallel with a multiply, increasing overall performance for certain DSP operations.

Consider a typical filtering operation where the filter is defined as in Equation 5-1.

$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k) \qquad \textit{Eqn. 5-1}$$

Here, the output y(i) is determined by past output values and past input values. This is the general form of an infinite impulse response (IIR) filter. A finite impulse response (FIR) filter can be obtained by setting coefficients a(k) to zero. In either case, the operations involved in computing such a filter are multiplies and product summing. To show this point, reduce Equation 5-1 to a simple, four-tap FIR filter, shown in Equation 5-2, in which the accumulated sum is a past data values and coefficients sum.

$$y(i) = \sum_{k=0}^{3} b(k)x(i-k) = b(0)x(i) + b(1)x(i-1) + b(2)x(i-2) + b(3)x(i-3) \qquad \textit{Eqn. 5-2}$$

5.2 Memory Map/Register Definition

The following table and sections explain the MAC registers:

Table 5-1. EMAC Memory Map

BDM[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	5.2.1/5-4
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	5.2.2/5-6
0x806	MAC Accumulator 0 (ACC0)	32	R/W	Undefined	5.2.3/5-8
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	5.2.4/5-8
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	5.2.4/5-8
0x809	MAC Accumulator 1 (ACC1)	32	R/W	Undefined	5.2.3/5-8
0x80A	MAC Accumulator 2 (ACC2)	32	R/W	Undefined	5.2.3/5-8
0x80B	MAC Accumulator 3 (ACC3)	32	R/W	Undefined	5.2.3/5-8

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 34, "Debug Module."

5.2.1 MAC Status Register (MACSR)

The MAC status register (MACSR) contains a 4-bit operational mode field and condition flags. Operational mode bits control whether operands are signed or unsigned and whether they are treated as integers or fractions. These bits also control the overflow/saturation mode and the way in which rounding is performed. Negative, zero, and multiple overflow condition flags are also provided.

BDM: 0x804 (MACSR)

Access: Supervisor read/write
BDM 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAV*n*				OMC	S/U	F/I	R/T	N	Z	V	EV
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5-2. MAC Status Register (MACSR)

Table 5-2. MACSR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11–8 PAV*n*	Product/accumulation overflow flags. Contains four flags, one per accumulator, that indicate if past MAC or MSAC instructions generated an overflow during product calculation or the 48-bit accumulation. When a MAC or MSAC instruction is executed, the PAV*n* flag associated with the destination accumulator forms the general overflow flag, MACSR[V]. Once set, each flag remains set until V is cleared by a `move.l, MACSR` instruction or the accumulator is loaded directly. Bit 11: Accumulator 3 ... Bit 8: Accumulator 0
7 OMC	Overflow saturation mode. Enables or disables saturation mode on overflow. If set, the accumulator is set to the appropriate constant (see S/U field description) on any operation that overflows the accumulator. After saturation, the accumulator remains unaffected by any other MAC or MSAC instructions until the overflow bit is cleared or the accumulator is directly loaded.
6 S/U	Signed/unsigned operations. **In integer mode:** S/U determines whether operations performed are signed or unsigned. It also determines the accumulator value during saturation, if enabled. 0 Signed numbers. On overflow, if OMC is enabled, an accumulator saturates to the most positive (0x7FFF_FFFF) or the most negative (0x8000_0000) number, depending on the instruction and the product value that overflowed. 1 Unsigned numbers. On overflow, if OMC is enabled, an accumulator saturates to the smallest value (0x0000_0000) or the largest value (0xFFFF_FFFF), depending on the instruction. **In fractional mode:** S/U controls rounding while storing an accumulator to a general-purpose register. 0 Move accumulator without rounding to a 16-bit value. Accumulator is moved to a general-purpose register as a 32-bit value. 1 The accumulator is rounded to a 16-bit value using the round-to-nearest (even) method when moved to a general-purpose register. See Section 5.3.1.1, “Rounding”. The resulting 16-bit value is stored in the lower word of the destination register. The upper word is zero-filled. This rounding procedure does not affect the accumulator value.
5 F/I	Fractional/integer mode. Determines whether input operands are treated as fractions or integers. 0 Integers can be represented in signed or unsigned notation, depending on the value of S/U. 1 Fractions are represented in signed, fixed-point, two’s complement notation. Values range from -1 to 1 - 2^{-15} for 16-bit fractions and -1 to 1 - 2^{-31} for 32-bit fractions. See Section 5.3.4, “Data Representation.”
4 R/T	Round/truncate mode. Controls rounding procedure for `move.l ACCx,Rx`, or MSAC.L instructions when in fractional mode. 0 Truncate. The product’s lsbs are dropped before it is combined with the accumulator. Additionally, when a store accumulator instruction is executed (`move.l ACCx,Rx`), the 8 lsbs of the 48-bit accumulator logic are truncated. 1 Round-to-nearest (even). The 64-bit product of two 32-bit, fractional operands is rounded to the nearest 40-bit value. If the low-order 24 bits equal 0x80_0000, the upper 40 bits are rounded to the nearest even (lsb = 0) value. See Section 5.3.1.1, “Rounding”. Additionally, when a store accumulator instruction is executed (`move.l ACCx,Rx`), the lsbs of the 48-bit accumulator logic round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared and MACSR[R/T] is set, the low-order 8 bits are used to round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.

Table 5-2. MACSR Field Descriptions (continued)

Field	Description
3 N	Negative. Set if the msb of the result is set, otherwise cleared. N is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
2 Z	Zero. Set if the result equals zero, otherwise cleared. This bit is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
1 V	Overflow. Set if an arithmetic overflow occurs on a MAC or MSAC instruction, indicating that the result cannot be represented in the limited width of the EMAC. V is set only if a product overflow occurs or the accumulation overflows the 48-bit structure. V is evaluated on each MAC or MSAC operation and uses the appropriate PAV*n* flag in the next-state V evaluation.
0 EV	Extension overflow. Signals that the last MAC or MSAC instruction overflowed the 32 lsbs in integer mode or the 40 lsbs in fractional mode of the destination accumulator. However, the result remains accurately represented in the combined 48-bit accumulator structure. Although an overflow has occurred, the correct result, sign, and magnitude are contained in the 48-bit accumulator. Subsequent MAC or MSAC operations may return the accumulator to a valid 32/40-bit result.

Table 5-3 summarizes the interaction of the MACSR[S/U,F/I,R/T] control bits.

Table 5-3. Summary of S/U, F/I, and R/T Control Bits

S/U	F/I	R/T	Operational Modes
0	0	x	Signed, integer
0	1	0	Signed, fractional Truncate on MAC.L and MSAC.L No round on accumulator stores
0	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-32-bits on accumulator stores
1	0	x	Unsigned, integer
1	1	0	Signed, fractional Truncate on MAC.L and MSAC.L Round-to-16-bits on accumulator stores
1	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-16-bits on accumulator stores

5.2.2 Mask Register (MASK)

The 32-bit MASK implements the low-order 16 bits to minimize the alignment complications involved with loading and storing only 16 bits. When the MASK is loaded, the low-order 16 bits of the source operand are actually loaded into the register. When it is stored, the upper 16 bits are all forced to ones.

This register performs a simple AND with the operand address for MAC instructions. The processor calculates the normal operand address and, if enabled, that address is then ANDed with {0xFFFF, MASK[15:0]} to form the final address. Therefore, with certain MASK bits cleared, the operand address can be constrained to a certain memory region. This is used primarily to implement circular queues with the (An)+ addressing mode.

This minimizes the addressing support required for filtering, convolution, or any routine that implements a data array as a circular queue. For MAC + MOVE operations, the MASK contents can optionally be included in all memory effective address calculations. The syntax is as follows:

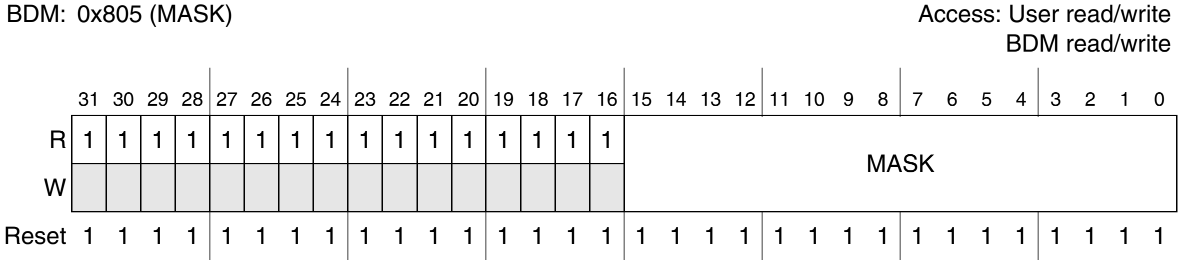
```
mac.sz  Ry,RxSF,<ea>y&,Rw
```

The & operator enables the MASK use and causes bit 5 of the extension word to be set. The exact algorithm for the use of MASK is:

```
if extension word, bit [5] = 1, the MASK bit, then
         if <ea> = (An)
                  oa  =  An & {0xFFFF, MASK}

         if <ea> = (An)+
                  oa  =  An
                  An  = (An + 4) & {0xFFFF, MASK}

         if <ea> =-(An)
                  oa  = (An - 4) & {0xFFFF, MASK}
                  An  = (An - 4) & {0xFFFF, MASK}

         if <ea> = (d16,An)
                  oa  = (An + se_d16) & {0xFFFF0x, MASK}
```

Here, *oa* is the calculated operand address and *se_d16* is a sign-extended 16-bit displacement. For auto-addressing modes of post-increment and pre-decrement, the updated An value calculation is also shown.

Use of the post-increment addressing mode, {(An)+} with the MASK is suggested for circular queue implementations.



BDM: 0x805 (MASK)

Access: User read/write
BDM 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
R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	MASK															
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 5-3. Mask Register (MASK)

Table 5-4. MASK Field Descriptions

Field	Description
31–16	Reserved, must be set.
15–0 MASK	Performs a simple AND with the operand address for MAC instructions.

5.2.3 Accumulator Registers (ACC0–3)

The accumulator registers store 32-bits of the MAC operation result. The accumulator extension registers form the entire 48-bit result.

BDM: 0x806 (ACC0)
0x809 (ACC1)
0x80A (ACC2)
0x80B (ACC3)

Access: User read/write
BDM 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
R	Accumulator																															
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 5-4. Accumulator Registers (ACC0–3)

Table 5-5. ACC0–3 Field Descriptions

Field	Description
31–0 Accumulator	Store 32-bits of the result of the MAC operation.

5.2.4 Accumulator Extension Registers (ACCext01, ACCext23)

Each pair of 8-bit accumulator extension fields are concatenated with the corresponding 32-bit accumulator register to form the 48-bit accumulator. For more information, see Section 5.3, “Functional Description.”

BDM: 0x807 (ACCext01)

Access: User read/write
BDM 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
R	ACC0U								ACC0L								ACC1U								ACC1L							
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 5-5. Accumulator Extension Register (ACCext01)

Table 5-6. ACCext01 Field Descriptions

Field	Description
31–24 ACC0U	Accumulator 0 upper extension byte
23–16 ACC0L	Accumulator 0 lower extension byte
15–8 ACC1U	Accumulator 1 upper extension byte
7–0 ACC1L	Accumulator 1 lower extension byte

BDM: 0x808 (ACCext23)

Access: User read/write
BDM 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
R W	ACC2U								ACC2L								ACC3U								ACC3L							
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 5-6. Accumulator Extension Register (ACCext23)

Table 5-7. ACCext23 Field Descriptions

Field	Description
31–24 ACC2U	Accumulator 2 upper extension byte
23–16 ACC2L	Accumulator 2 lower extension byte
15–8 ACC3U	Accumulator 3 upper extension byte
7–0 ACC3L	Accumulator 3 lower extension byte

5.3 Functional Description

The MAC speeds execution of ColdFire integer-multiply instructions (MULS and MULU) and provides additional functionality for multiply-accumulate operations. By executing MULS and MULU in the MAC, execution times are minimized and deterministic compared to the 2-bit/cycle algorithm with early termination that the OEP normally uses if no MAC hardware is present.

The added MAC instructions to the ColdFire ISA provide for the multiplication of two numbers, followed by the addition or subtraction of the product to or from the value in an accumulator. Optionally, the product may be shifted left or right by 1 bit before addition or subtraction. Hardware support for saturation arithmetic can be enabled to minimize software overhead when dealing with potential overflow conditions. Multiply-accumulate operations support 16- or 32-bit input operands in these formats:

- Signed integers
- Unsigned integers
- Signed, fixed-point, fractional numbers

The EMAC is optimized for single-cycle, pipelined 32 × 32 multiplications. For word- and longword-sized integer input operands, the low-order 40 bits of the product are formed and used with the destination accumulator. For fractional operands, the entire 64-bit product is calculated and truncated or rounded to the most-significant 40-bit result using the round-to-nearest (even) method before it is combined with the destination accumulator.

For all operations, the resulting 40-bit product is extended to a 48-bit value (using sign-extension for signed integer and fractional operands, zero-fill for unsigned integer operands) before being combined with the 48-bit destination accumulator.

Figure 5-7 and Figure 5-8 show relative alignment of input operands, the full 64-bit product, the resulting 40-bit product used for accumulation, and 48-bit accumulator formats.

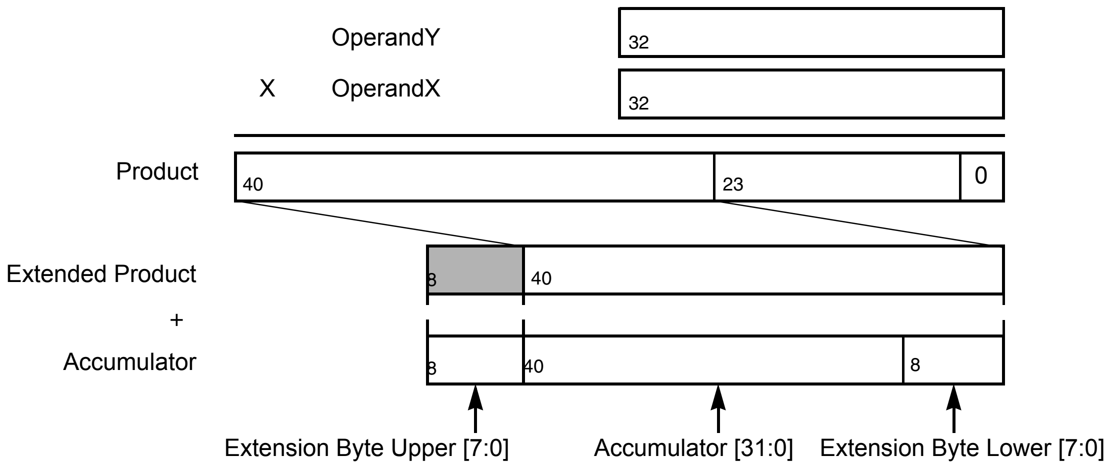


Figure 5-7. Fractional Alignment

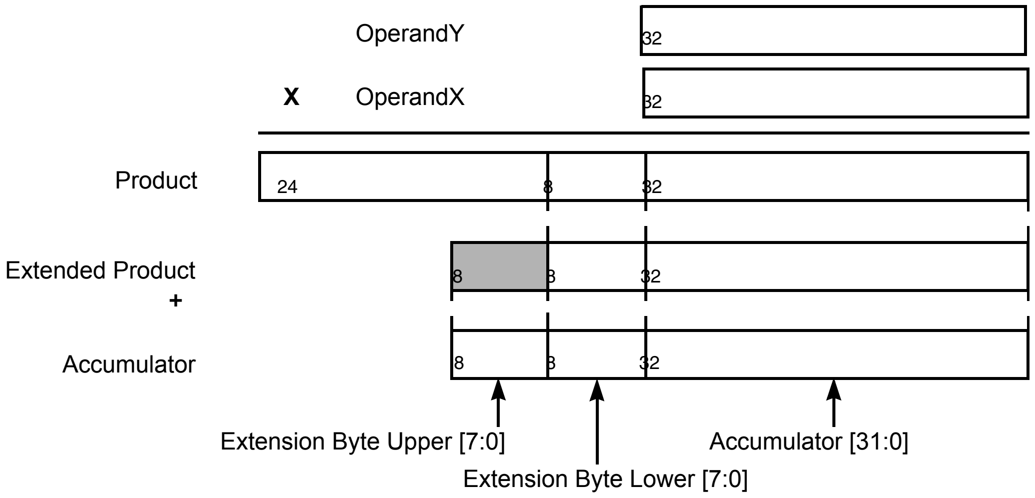


Figure 5-8. Signed and Unsigned Integer Alignment

Therefore, the 48-bit accumulator definition is a function of the EMAC operating mode. Given that each 48-bit accumulator is the concatenation of 16-bit accumulator extension register (ACCext*n*) contents and 32-bit ACC*n* contents, the specific definitions are:

```
if MACSR[6:5] == 00        /* signed integer mode */
      Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}
if MACSR[6:5] == 01 or 11  /* signed fractional mode */
      Complete Accumulator [47:0] = {ACCextn[15:8], ACCn[31:0], ACCextn[7:0]}
if MACSR[6:5] == 10        /* unsigned integer mode */
      Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}
```

The four accumulators are represented as an array, ACC*n*, where *n* selects the register.

Although the multiplier array is implemented in a four-stage pipeline, all arithmetic MAC instructions have an effective issue rate of 1 cycle, regardless of input operand size or type.

All arithmetic operations use register-based input operands, and summed values are stored in an accumulator. Therefore, an additional MOVE instruction is needed to store data in a general-purpose register. One new feature in EMAC instructions is the ability to choose the upper or lower word of a register as a 16-bit input operand. This is useful in filtering operations if one data register is loaded with the input data and another is loaded with the coefficient. Two 16-bit multiply accumulates can be performed without fetching additional operands between instructions by alternating word choice during calculations.

The EMAC has four accumulator registers versus the MAC's single accumulator. The additional registers improve the performance of some algorithms by minimizing pipeline stalls needed to store an accumulator value back to general-purpose registers. Many algorithms require multiple calculations on a given data set. By applying different accumulators to these calculations, it is often possible to store one accumulator without any stalls while performing operations involving a different destination accumulator.

The need to move large amounts of data presents an obstacle to obtaining high throughput rates in DSP engines. Existing ColdFire instructions can accommodate these requirements. A MOVEM instruction can efficiently move large data blocks by generating line-sized burst references. The ability to load an operand simultaneously from memory into a register and execute a MAC instruction makes some DSP operations such as filtering and convolution more manageable.

The programming model includes a mask register (MASK), which can optionally be used to generate an operand address during MAC + MOVE instructions. The register application with auto-increment addressing mode supports efficient implementation of circular data queues for memory operands.

5.3.1 Fractional Operation Mode

This section describes behavior when the fractional mode is used (MACSR[F/I] is set).

5.3.1.1 Rounding

When the processor is in fractional mode, there are two operations during which rounding can occur:

1. Execution of a store accumulator instruction (`move.l ACCx,Rx`). The lsbs of the 48-bit accumulator logic are used to round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared, the low-order 8 bits round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.
2. Execution of a MAC (or MSAC) instruction with 32-bit operands. If MACSR[R/T] is zero, multiplying two 32-bit numbers creates a 64-bit product truncated to the upper 40 bits; otherwise, it is rounded using round-to-nearest (even) method.

To understand the round-to-nearest-even method, consider the following example involving the rounding of a 32-bit number, R0, to a 16-bit number. Using this method, the 32-bit number is rounded to the closest 16-bit number possible. Let the high-order 16 bits of R0 be named R0.U and the low-order 16 bits be R0.L.

- If R0.L is less than 0x8000, the result is truncated to the value of R0.U.
- If R0.L is greater than 0x8000, the upper word is incremented (rounded up).

- If R0.L is 0x8000, R0 is half-way between two 16-bit numbers. In this case, rounding is based on the lsb of R0.U, so the result is always even (lsb = 0).
  - If the lsb of R0.U equals 1 and R0.L equals 0x8000, the number is rounded up.
  - If the lsb of R0.U equals 0 and R0.L equals 0x8000, the number is rounded down.

This method minimizes rounding bias and creates as statistically correct an answer as possible.

The rounding algorithm is summarized in the following pseudocode:

```
if R0.L < 0x8000
         then Result = R0.U
else if R0.L > 0x8000
         then Result = R0.U + 1
else if lsb of R0.U = 0             /* R0.L = 0x8000 */
         then Result = R0.U
else Result = R0.U + 1
```

The round-to-nearest-even technique is also known as convergent rounding.

5.3.1.2 Saving and Restoring the EMAC Programming Model

The presence of rounding logic in the EMAC output datapath requires special care during the EMAC's save/restore process. In particular, any result rounding modes must be disabled during the save/restore process so the exact bit-wise contents of the EMAC registers are accessed. Consider the memory structure containing the EMAC programming model:

```
struct   macState {
         int acc0;
         int acc1;
         int acc2;
         int acc3;
         int accext01;
         int accext02;
         int mask;
         int macsr;
} macState;
```

The following assembly language routine shows the proper sequence for a correct EMAC state save. This code assumes all Dn and An registers are available for use, and the memory location of the state save is defined by A7.

```
EMAC_state_save:
         move.l  macsr,d7             ; save the macsr
         clr.l   d0                   ; zero the register to ...
         move.l  d0,macsr             ; disable rounding in the macsr
         move.l  acc0,d0              ; save the accumulators
         move.l  acc1,d1
         move.l  acc2,d2
         move.l  acc3,d3
         move.l  accext01,d4          ; save the accumulator extensions
         move.l  accext23,d5
         move.l  mask,d6              ; save the address mask
         movem.l #0x00ff,(a7)         ; move the state to memory
```

This code performs the EMAC state restore:

```
EMAC_state_restore:
        movem.l  (a7),#0x00ff       ; restore the state from memory
        move.l   #0,macsr           ; disable rounding in the macsr
        move.l   d0,acc0            ; restore the accumulators
        move.l   d1,acc1
        move.l   d2,acc2
        move.l   d3,acc3
        move.l   d4,accext01        ; restore the accumulator extensions
        move.l   d5,accext23
        move.l   d6,mask            ; restore the address mask
        move.l   d7,macsr           ; restore the macsr
```

Executing this sequence type can correctly save and restore the exact state of the EMAC programming model.

5.3.1.3 MULS/MULU

MULS and MULU are unaffected by fractional-mode operation; operands remain assumed to be integers.

5.3.1.4 Scale Factor in MAC or MSAC Instructions

The scale factor is ignored while the MAC is in fractional mode.

5.3.2 EMAC Instruction Set Summary

Table 5-8 summarizes EMAC unit instructions.

Table 5-8. EMAC Instruction Summary

Command	Mnemonic	Description
Multiply Signed	`muls <ea>y,Dx`	Multiplies two signed operands yielding a signed result
Multiply Unsigned	`mulu <ea>y,Dx`	Multiplies two unsigned operands yielding an unsigned result

Multiply Accumulate	`mac Ry,RxSF,ACCx`
	`msac Ry,RxSF,ACCx`
		Multiplies two operands and adds/subtracts the product to/from an accumulator

Multiply Accumulate with Load	`mac  Ry,Rx,<ea>y,Rw,ACCx`
	`msac Ry,Rx,<ea>y,Rw,ACCx`
		Multiplies two operands and combines the product to an accumulator while loading a register with the memory operand

Load Accumulator	`move.l {Ry,#imm},ACCx`	Loads an accumulator with a 32-bit operand
Store Accumulator	`move.l ACCx,Rx`	Writes the contents of an accumulator to a CPU register
Copy Accumulator	`move.l ACCy,ACCx`	Copies a 48-bit accumulator
Load MACSR	`move.l {Ry,#imm},MACSR`	Writes a value to MACSR
Store MACSR	`move.l MACSR,Rx`	Write the contents of MACSR to a CPU register
Store MACSR to CCR	`move.l MACSR,CCR`	Write the contents of MACSR to the CCR
Load MAC Mask Reg	`move.l {Ry,#imm},MASK`	Writes a value to the MASK register
Store MAC Mask Reg	`move.l MASK,Rx`	Writes the contents of the MASK to a CPU register
Load Accumulator Extensions 01	`move.l {Ry,#imm},ACCext01`	Loads the accumulator 0,1 extension bytes with a 32-bit operand

Table 5-8. EMAC Instruction Summary (continued)

Command	Mnemonic	Description
Load Accumulator Extensions 23	`move.l {Ry,#imm},ACCext23`	Loads the accumulator 2,3 extension bytes with a 32-bit operand
Store Accumulator Extensions 01	`move.l ACCext01,Rx`	Writes the contents of accumulator 0,1 extension bytes into a CPU register
Store Accumulator Extensions 23	`move.l ACCext23,Rx`	Writes the contents of accumulator 2,3 extension bytes into a CPU register

5.3.3 EMAC Instruction Execution Times

The instruction execution times for the EMAC can be found in Section 3.3.5.6, “EMAC Instruction Execution Times”.

The EMAC execution pipeline overlaps the EX stage of the OEP (the first stage of the EMAC pipeline is the last stage of the basic OEP). EMAC units are designed for sustained, fully-pipelined operation on accumulator load, copy, and multiply-accumulate instructions. However, instructions that store contents of the multiply-accumulate programming model can generate OEP stalls that expose the EMAC execution pipeline depth:

```
mac.w    Ry, Rx, Acc0
move.l   Acc0, Rz
```

The MOVE.L instruction that stores the accumulator to an integer register (Rz) stalls until the program-visible copy of the accumulator is available. Figure 5-9 shows EMAC timing.

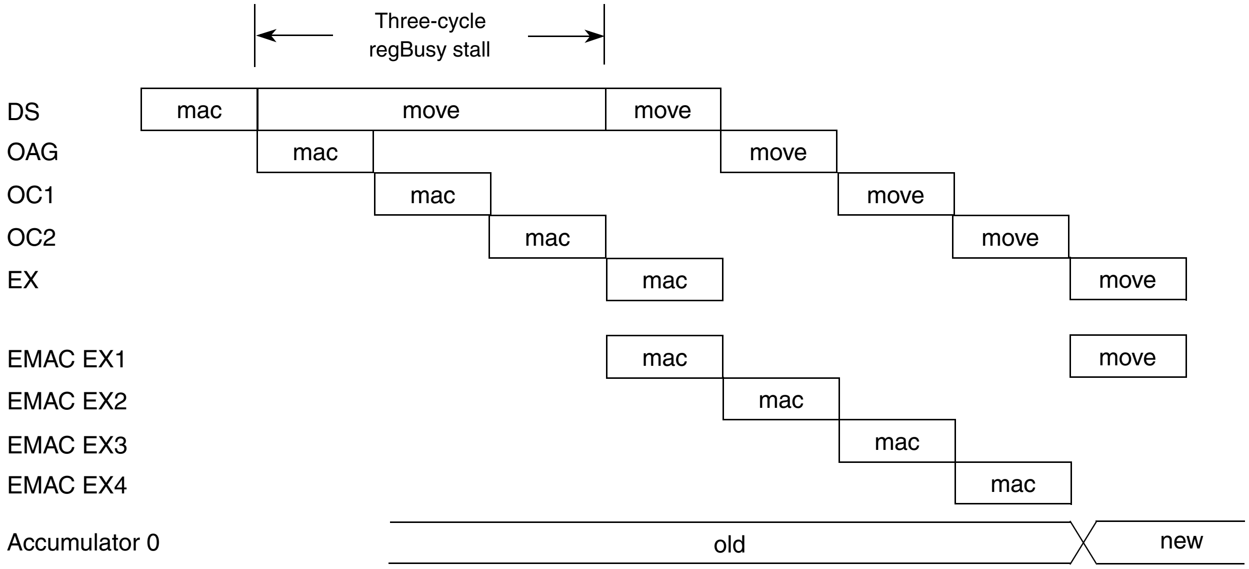


Figure 5-9. EMAC-Specific OEP Sequence Stall

In Figure 5-9, the OEP stalls the store-accumulator instruction for three cycles: the EMAC pipeline depth minus 1. The minus 1 factor is needed because the OEP and EMAC pipelines overlap by a cycle, the EX stage. As the store-accumulator instruction reaches the EX stage where the operation is performed, the recently-updated accumulator 0 value is available.

As with change or use stalls between accumulators and general-purpose registers, introducing intervening instructions that do not reference the busy register can reduce or eliminate sequence-related store-MAC instruction stalls. A major benefit of the EMAC is the addition of three accumulators to minimize stalls caused by exchanges between accumulator(s) and general-purpose registers.

5.3.4 Data Representation

MACSR[S/U,F/I] selects one of the following three modes, where each mode defines a unique operand type:

1. Two’s complement signed integer: In this format, an N-bit operand value lies in the range $-2^{(N-1)} \leq \text{operand} \leq 2^{(N-1)} - 1$. The binary point is right of the lsb.
2. Unsigned integer: In this format, an N-bit operand value lies in the range $0 \leq \text{operand} \leq 2^{N} - 1$. The binary point is right of the lsb.
3. Two’s complement, signed fractional: In an N-bit number, the first bit is the sign bit. The remaining bits signify the first N-1 bits after the binary point. Given an N-bit number, $a_{N-1}a_{N-2}a_{N-3}\ldots a_2a_1a_0$, its value is given by the equation in Equation 5-3.

$$\text{value} = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{-(i+1-N)} \cdot ai \qquad \textbf{Eqn. 5-3}$$

This format can represent numbers in the range $-1 \leq \text{operand} \leq 1 - 2^{-(N-1)}$.

For words and longwords, the largest negative number that can be represented is -1, whose internal representation is 0x8000 and 0x8000_0000, respectively. The largest positive word is 0x7FFF or $(1 - 2^{-15})$; the most positive longword is 0x7FFF_FFFF or $(1 - 2^{-31})$. Thus, the number range for these signed fractional numbers is [-1.0, ..., 1.0].

5.3.5 MAC Opcodes

MAC opcodes are described in the *ColdFire Programmer’s Reference Manual*.

Remember the following:

- Unless otherwise noted, the value of MACSR[N,Z] is based on the result of the final operation that involves the product and the accumulator.
- The overflow (V) flag is managed differently. It is set if the complete product cannot be represented as a 40-bit value (this applies to 32 × 32 integer operations only) or if the combination of the product with an accumulator cannot be represented in the given number of bits. The EMAC design includes an additional product/accumulation overflow bit for each accumulator that are treated as sticky indicators and are used to calculate the V bit on each MAC or MSAC instruction. See Section 5.2.1, “MAC Status Register (MACSR)”.

- For the MAC design, the assembler syntax of the MAC (multiply and add to accumulator) and MSAC (multiply and subtract from accumulator) instructions does not include a reference to the single accumulator. For the EMAC, assemblers support this syntax and no explicit reference to an accumulator is interpreted as a reference to ACC0. Assemblers also support syntaxes where the destination accumulator is explicitly defined.
- The optional 1-bit shift of the product is specified using the notation {<< | >>} SF, where <<1 indicates a left shift and >>1 indicates a right shift. The shift is performed before the product is added to or subtracted from the accumulator. Without this operator, the product is not shifted. If the EMAC is in fractional mode (MACSR[F/I] is set), SF is ignored and no shift is performed. Because a product can overflow, the following guidelines are implemented:
  - For unsigned word and longword operations, a zero is shifted into the product on right shifts.
  - For signed, word operations, the sign bit is shifted into the product on right shifts unless the product is zero. For signed, longword operations, the sign bit is shifted into the product unless an overflow occurs or the product is zero, in which case a zero is shifted in.
  - For all left shifts, a zero is inserted into the lsb position.

The following pseudocode explains basic MAC or MSAC instruction functionality. This example is presented as a case statement covering the three basic operating modes with signed integers, unsigned integers, and signed fractionals. Throughout this example, a comma-separated list in curly brackets, {}, indicates a concatenation operation.

```
switch (MACSR[6:5])     /* MACSR[S/U, F/I] */
{
   case 0:              /* signed integers */
     if (MACSR.OMC == 0 || MACSR.PAVn == 0)
         then {
               MACSR.PAVn = 0
               /* select the input operands */
               if (sz == word)
                  then {if (U/Ly == 1)
                       then operandY[31:0] = {sign-extended Ry[31], Ry[31:16]}
                        else operandY[31:0] = {sign-extended Ry[15], Ry[15:0]}
                        if (U/Lx == 1)
                       then operandX[31:0] = {sign-extended Rx[31], Rx[31:16]}
                        else operandX[31:0] = {sign-extended Rx[15], Rx[15:0]}
                  }
                  else {operandY[31:0] = Ry[31:0]
                        operandX[31:0] = Rx[31:0]
                  }

               /* perform the multiply */
               product[63:0] = operandY[31:0] * operandX[31:0]

               /* check for product overflow */
       if ((product[63:39] != 0x0000_00_0) && (product[63:39] != 0xffff_ff_1))
                  then {        /* product overflow */
                        MACSR.PAVn = 1
                        MACSR.V = 1
                        if (inst == MSAC && MACSR.OMC == 1)
                           then if (product[63] == 1)
                                   then result[47:0] = 0x0000_7fff_ffff
                                   else result[47:0] = 0xffff_8000_0000
                           else if (MACSR.OMC == 1)
                                   then /* overflowed MAC,
                                           saturationMode enabled */
                                        if (product[63] == 1)
                                          then result[47:0] = 0xffff_8000_0000
                                          else result[47:0] = 0x0000_7fff_ffff
                  }

               /* sign-extend to 48 bits before performing any scaling */
                        product[47:40] = {8{product[39]}}    /* sign-extend */

               /* scale product before combining with accumulator */
               switch (SF)     /* 2-bit scale factor */
               {
                   case 0:     /* no scaling specified */
                      break;
                   case 1:     /* SF = "<< 1" */
                      product[40:0] = {product[39:0], 0}
                      break;
                   case 2:     /* reserved encoding */
                      break;
                   case 3:     /* SF = ">> 1" */
                      product[39:0] = {product[39], product[39:1]}
                      break;
               }

               if (MACSR.PAVn == 0)
                  then {if (inst == MSAC)
                           then result[47:0] = ACCx[47:0] - product[47:0]
                           else result[47:0] = ACCx[47:0] + product[47:0]
                  }

               /* check for accumulation overflow */
               if (accumulationOverflow == 1)
                  then {MACSR.PAVn = 1
                        MACSR.V = 1
                        if (MACSR.OMC == 1)
                           then /* accumulation overflow,
                                   saturationMode enabled */
                                if (result[47] == 1)
                                   then result[47:0] = 0x0000_7fff_ffff
                                   else result[47:0] = 0xffff_8000_0000
                  }
               /* transfer the result to the accumulator */
               ACCx[47:0] = result[47:0]
         }
         MACSR.V = MACSR.PAVn
         MACSR.N = ACCx[47]
         if (ACCx[47:0] == 0x0000_0000_0000)
            then MACSR.Z = 1
            else MACSR.Z = 0
         if ((ACCx[47:31] == 0x0000_0) || (ACCx[47:31] == 0xffff_1))
            then MACSR.EV = 0
            else MACSR.EV = 1
   break;
      case 1,3:              /* signed fractionals */
      if (MACSR.OMC == 0 || MACSR.PAVn == 0)
```

```
      then {
            MACSR.PAVn = 0
            if (sz == word)
               then {if (U/Ly == 1)
                        then operandY[31:0] = {Ry[31:16], 0x0000}
                        else operandY[31:0] = {Ry[15:0],  0x0000}
                     if (U/Lx == 1)
                        then operandX[31:0] = {Rx[31:16], 0x0000}
                        else operandX[31:0] = {Rx[15:0],  0x0000}
               }
               else {operandY[31:0] = Ry[31:0]
                     operandX[31:0] = Rx[31:0]
               }
            /* perform the multiply */
            product[63:0] = (operandY[31:0] * operandX[31:0]) << 1
            /* check for product rounding */
            if (MACSR.R/T == 1)
               then { /* perform convergent rounding */
                     if (product[23:0] > 0x80_0000)
                        then product[63:24] = product[63:24] + 1
               else if ((product[23:0] == 0x80_0000) && (product[24] == 1))
                                then product[63:24] = product[63:24] + 1
               }
            /* sign-extend to 48 bits and combine with accumulator */
            /* check for the -1 * -1 overflow case */
    if ((operandY[31:0] == 0x8000_0000) && (operandX[31:0] == 0x8000_0000))
               then product[71:64] = 0x00                 /* zero-fill */
               else product[71:64] = {8{product[63]}}    /* sign-extend */
            if (inst == MSAC)
               then result[47:0] = ACCx[47:0] - product[71:24]
               else result[47:0] = ACCx[47:0] + product[71:24]
            /* check for accumulation overflow */
            if (accumulationOverflow == 1)
               then {MACSR.PAVn = 1
                     MACSR.V = 1
                     if (MACSR.OMC == 1)
                        then /* accumulation overflow,
                                saturationMode enabled */
                             if (result[47] == 1)
                                then result[47:0] = 0x007f_ffff_ff00
                                else result[47:0] = 0xff80_0000_0000
               }
            /* transfer the result to the accumulator */
            ACCx[47:0] = result[47:0]
               }
      MACSR.V = MACSR.PAVn
      MACSR.N = ACCx[47]
      if (ACCx[47:0] == 0x0000_0000_0000)
          then MACSR.Z = 1
          else MACSR.Z = 0
      if ((ACCx[47:39] == 0x00_0) || (ACCx[47:39] == 0xff_1))
          then MACSR.EV = 0
          else MACSR.EV = 1
break;
case 2:              /* unsigned integers */
   if (MACSR.OMC == 0 || MACSR.PAVn == 0)
      then {
```

```
MACSR.PAVn = 0
/* select the input operands */
if (sz == word)
   then {if (U/Ly == 1)
            then operandY[31:0] = {0x0000, Ry[31:16]}
            else operandY[31:0] = {0x0000, Ry[15:0]}
         if (U/Lx == 1)
            then operandX[31:0] = {0x0000, Rx[31:16]}
            else operandX[31:0] = {0x0000, Rx[15:0]}
   }
   else {operandY[31:0] = Ry[31:0]
         operandX[31:0] = Rx[31:0]
   }

/* perform the multiply */
product[63:0] = operandY[31:0] * operandX[31:0]

/* check for product overflow */
if (product[63:40] != 0x0000_00)
   then {         /* product overflow */
         MACSR.PAVn = 1
         MACSR.V = 1
         if (inst == MSAC && MACSR.OMC == 1)
            then result[47:0] = 0x0000_0000_0000
            else if (MACSR.OMC == 1)
                    then /* overflowed MAC,
                            saturationMode enabled */
                         result[47:0] = 0xffff_ffff_ffff
   }

/* zero-fill to 48 bits before performing any scaling */
         product[47:40] = 0    /* zero-fill upper byte */

/* scale product before combining with accumulator */
switch (SF)     /* 2-bit scale factor */
{
    case 0:     /* no scaling specified */
       break;
    case 1:     /* SF = "<< 1" */
       product[40:0] = {product[39:0], 0}
       break;
    case 2:     /* reserved encoding */
       break;
    case 3:     /* SF = ">> 1" */
       product[39:0] = {0, product[39:1]}
       break;
}

/* combine with accumulator */
if (MACSR.PAVn == 0)
   then {if (inst == MSAC)
            then result[47:0] = ACCx[47:0] - product[47:0]
            else result[47:0] = ACCx[47:0] + product[47:0]
   }

/* check for accumulation overflow */
if (accumulationOverflow == 1)
```

```
                  then {MACSR.PAVn = 1
                        MACSR.V = 1
                        if (inst == MSAC && MACSR.OMC == 1)
                           then result[47:0] = 0x0000_0000_0000
                           else if (MACSR.OMC == 1)
                                then /* overflowed MAC,
                                        saturationMode enabled */
                                     result[47:0] = 0xffff_ffff_ffff
                  }

               /* transfer the result to the accumulator */
               ACCx[47:0] = result[47:0]
         }
         MACSR.V = MACSR.PAVn
         MACSR.N = ACCx[47]
         if (ACCx[47:0] == 0x0000_0000_0000)
            then MACSR.Z = 1
            else MACSR.Z = 0
         if (ACCx[47:32] == 0x0000)
            then MACSR.EV = 0
            else MACSR.EV = 1
   break;
}
```

Chapter 6
Cache

6.1 Introduction

This section describes the cache module, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

6.1.1 Block Diagram

Figure 6-1 shows the organization and integration of the data cache.

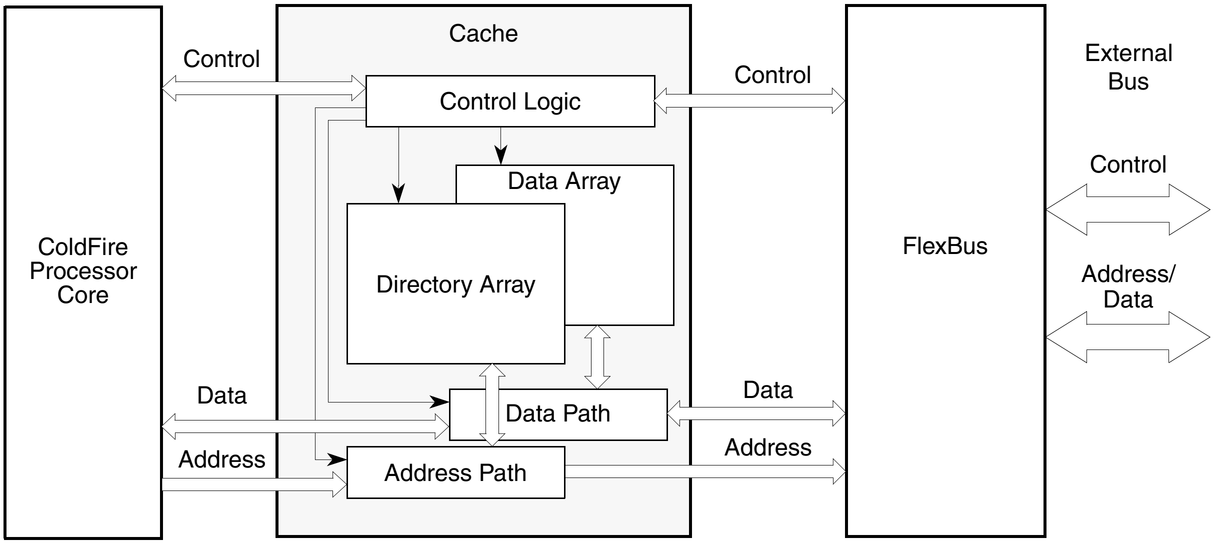


Figure 6-1. Data Cache Organization

6.1.2 Overview

The processor’s memory structure includes a 16-Kbyte data cache and a 16-Kbyte instruction cache. Both are non-blocking and four-way set-associative with a 16-byte line size. The cache improves system performance by providing single-cycle access to the instruction and data pipelines. This decouples processor performance from system-memory performance, increasing bus availability for on-chip DMA or external devices.

This device implements a special branch instruction cache for accelerating branches, enabled by a bit in the cache access control register (CACR[BEC]). The branch cache is described in Section 3.1.1.1, “Change-of-Flow Acceleration.”

Instruction and data caches implement line-fill buffers to optimize line-sized burst accesses. The data cache supports operation of copyback, write-through, or cache-inhibited modes. A four-entry, 32-bit buffer supports cache line-push operations, and can be configured to defer write buffering in write-through or cache-inhibited modes. The cache lock feature can be used to guarantee deterministic response for critical code or data areas.

A non-blocking cache services read or write hits from the processor while a fill (caused by a cache allocation) is in progress. As Figure 6-1 shows, accesses use a single bus connected to the cache.

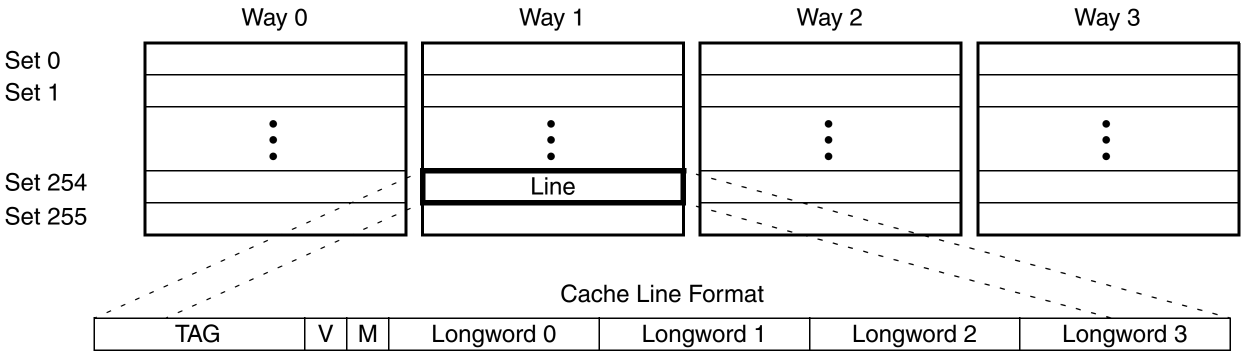
All addresses from the processor to the cache are physical addresses. A cache hit occurs when an address matches a cache entry. For a read, the cache supplies data to the processor. For a write, which is permitted to the data cache only, the processor updates the cache. If an access does not match a cache entry (misses the cache) or if a write access must be written through to memory, the cache performs a bus cycle on the internal bus and correspondingly on the external bus.

The cache module does not implement bus snooping; cache coherency with other possible bus masters must be maintained in software.

6.2 Cache Organization

A four-way set-associative cache is organized as four ways (levels). There are 256 sets in the 16-Kbyte data cache with each set defined as the grouping of four lines (one from each level, or way), corresponding to the same index into the cache array. Each line contains 16 bytes (4 longwords). The 16-Kbyte instruction cache has 256 sets as well. Entire cache lines are loaded from memory by burst-mode accesses that cache four longwords of data or instructions. All four longwords must be loaded for the cache line to be valid.

Figure 6-2 shows data cache organization, as well as terminology used.



Where:
TAG—20-bit address tag
V—Valid bit for line
M—Modified bit for line (data cache only)

Figure 6-2. Data Cache Organization and Line Format

6.2.1 Cache Line States: Invalid, Valid-Unmodified, and Valid-Modified

As shown in Table 6-1, a data cache line is always in one of three states: invalid, valid-unmodified (often referred to as exclusive), or valid-modified. An instruction cache line can be valid or invalid. A valid line can be explicitly invalidated by executing a CPUSHL instruction**.**

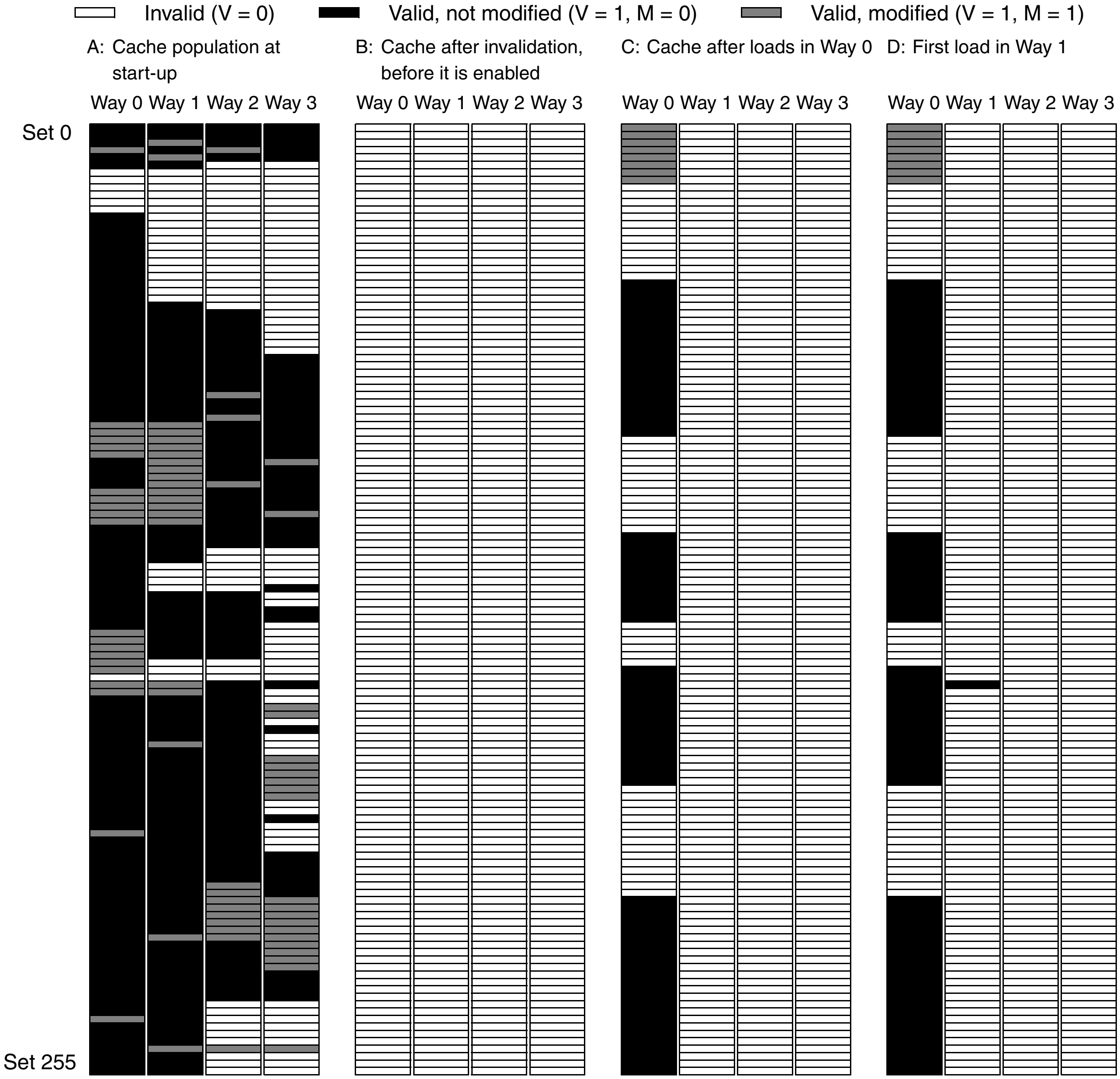
Table 6-1. Valid and Modified Bit Settings

V	M	Description
0	*x*	Invalid. Ignored during lookups.
1	0	Valid, unmodified. Cache line has valid data that matches system memory.
1	1	Valid, modified. Cache line contains most recent data, data at system memory location is stale.

6.2.2 The Cache at Start-Up

As Figure 6-3 (A) shows, after power-up, cache contents are undefined; V and M may be set on some lines even though the cache may not contain the appropriate data for start up. Because reset and power-up do not invalidate cache lines automatically, the cache should be cleared explicitly by setting CACR[DCINVA,ICINVA] before the cache is enabled (B).

After the entire cache is flushed, cacheable entries are loaded first in way 0. If way 0 is occupied, the cacheable entry is loaded into the same set in way 1, as shown in Figure 6-3 (D). This process is described in detail in Section 6.4, “Functional Description.”


Invalid (V = 0)
Valid, not modified (V = 1, M = 0)
Valid, modified (V = 1, M = 1)
Way 0 Way 1 Way 2 Way 3
Set 0
Set 255

A: Cache population at start-up

B: Cache after invalidation, before it is enabled

C: Cache after loads in Way 0

D: First load in Way 1

At reset, cache contents are indeterminate; V and M may be set. The cache should be cleared explicitly by setting CACR[DCINVA] before the cache is enabled.

Setting CACR[DCINVA] invalidates the entire cache.

Initial cacheable accesses to memory-fill positions in way 0.

A line is loaded in way 1 only if that set is full in way 0.

Figure 6-3. Data Cache: A) at Reset; B) after Invalidation; C and D) Loading Pattern

6.3 Memory Map/Register Definition

This section describes the implementation of the cache registers.

Table 6-2. Cache Memory Map

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	6.3.1/6-5
0x004	Access Control Register 0 (ACR0)	32	R/W	Undefined	Yes	6.3.2/6-8
0x005	Access Control Register 1 (ACR1)	32	R/W	Undefined	Yes	6.3.2/6-8
0x006	Access Control Register 2 (ACR2)	32	R/W	Undefined	Yes	6.3.2/6-8
0x007	Access Control Register 3 (ACR3)	32	R/W	Undefined	Yes	6.3.2/6-8

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 34, "Debug Module".

6.3.1 Cache Control Register (CACR)

The CACR register contains bits for configuring the cache. It can be written by the MOVEC register instruction and can be read or written from the debug facility. A hardware reset clears CACR, which disables the cache; however, reset does not affect tags, state information, or data in the cache.

BDM: 0x002

Access: MOVEC write-only
Debug read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DEC	DW	DESB	DDPI	DHLCK	DDCM		DC INVA	DDSP	0	0	IVO	BEC	BC INVA	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IEC	SPA	DNFB	IDPI	IHLCK	IDCM	0	IC INVA	IDSP	0	EUSP	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6-4. Cache-Control Register (CACR)

Table 6-3. CACR Field Descriptions

Field	Description
31 DEC	Enable data cache. 0 Cache disabled. The data cache is not operational, but data and tags are preserved. 1 Cache enabled.
30 DW	Data default write-protect. For normal operations that do not hit in the RAMBARs or ACRs, this field defines write-protection. See Section 6.4.1, "Caching Modes." 0 Not write protected. 1 Write protected. Write operations cause an access error exception.

Table 6-3. CACR Field Descriptions (continued)

Field	Description
29 DESB	Enable data store buffer. Affects the precision of transfers. 0 Imprecise-mode, write-through or cache-inhibited writes bypass the store buffer and generate bus cycles directly. Section 6.4.4.2.1, "Push and Store Buffers," describes the associated performance penalty. 1 The four-entry FIFO store buffer is enabled; to maximize performance, this buffer defers pending imprecise-mode, write-through or cache-inhibited writes. Precise-mode, cache-inhibited accesses always bypass the store buffer. Precise and imprecise modes are described in Section 6.4.1.2, "Cache-Inhibited Accesses."
28 DDPI	Data disable CPUSHL invalidate. 0 Normal operation. A CPUSHL instruction causes the selected line to be pushed if modified, then invalidated. 1 No clear operation. A CPUSHL instruction causes the selected line to be pushed if modified, then left valid.
27 DHLCK	Data cache half-data lock. 0 Normal operation. The cache allocates the lowest invalid way. If all ways are valid, the cache allocates the way pointed at by the round-robin counter and then increments this counter. 1 Half-cache operation. The cache allocates to the lower invalid way of levels 2 and 3; if both are valid, the cache allocates to Way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates Way 3 and increments the round-robin counter. This locks the contents of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions.
26–25 DDCM	Default data-cache mode. For normal operations that do not hit in the RAMBARs or ACRs, this field defines the effective cache mode. 00 Cacheable write-through imprecise 01 Cacheable copyback 10 Cache-inhibited precise 11 Cache-inhibited imprecise Precise and imprecise accesses are described in Section 6.4.1.2, "Cache-Inhibited Accesses."
24 DCINVA	Data cache invalidate all. Setting this bit initiates entire cache invalidation. After invalidation is complete, this bit automatically clears; it is not necessary to clear it explicitly. The caches are not cleared on power-up or normal reset, as shown in Figure 6-3. 0 No invalidation is performed. 1 Initiate invalidation of the entire data cache. The cache controller sequentially clears V and M bits in all sets. Subsequent data accesses stall until the invalidation is finished, at which point, this bit is automatically cleared. In copyback mode, the cache should be flushed using a CPUSHL instruction before setting this bit.
23 DDSP	Data default supervisor-protect. For normal operations that do not hit in the RAMBAR or ACRs, this field defines supervisor-protection 0 Not supervisor protected 1 Supervisor protected. User operations cause a fault
22–21	Reserved, must be cleared.
20 IVO	Invalidate only. Setting this bit forces the invalidation of only the referenced cache line when a CPUSHL instruction executes. See Section 6.4.8, "CPUSHL Enhancements," for more information.
19 BEC	Enable branch cache. 0 Branch cache disabled. This may be useful if code is unlikely to be reused. 1 Branch cache enabled.
18 BCINVA	Branch cache invalidate all. Invalidation occurs when this bit is set. Branch caches are not cleared on power-up or normal reset. 0 No invalidation is performed. 1 Initiate an invalidation of the entire branch cache.
17–16	Reserved, must be cleared.

Table 6-3. CACR Field Descriptions (continued)

Field	Description
15 IEC	Enable instruction cache 0 Instruction cache disabled. All instructions and tags in the cache are preserved. 1 Instruction cache enabled.
14 SPA	Search by physical address. Setting this bit forces the cache to search the accessed set with the supplied physical cache address when a CPUSHL instruction is executed. See Section 6.4.8, “CPUSHL Enhancements,” for more information.
13 DNFB	Default cache-inhibited fill buffer 0 Fill buffer does not store cache-inhibited instruction accesses (16 or 32 bits). 1 Fill buffer can store cache-inhibited accesses. The buffer is used only for normal (TT = 0) instruction reads of a cache-inhibited region. Instructions are loaded into the buffer by a burst access (line fill). They stay in the buffer until they are displaced; subsequent accesses may not appear on the external bus. Setting DNFB can cause a coherency problem for self-modifying code. If a cache-inhibited access uses the buffer while DNFB is set, instructions remain valid in the buffer until a cache-invalidate-all instruction, another cache-inhibited burst, or a miss that initiates a fill. A write to the line in the fill goes to the external bus without updating or invalidating the buffer. Subsequent reads are serviced by the fill buffer and receive stale information. **Note:** Freescale discourages the use of self-modifying code.
12 IDPI	Instruction CPUSHL invalidate disable. 0 Normal operation. A CPUSHL instruction invalidates the selected line. 1 No clear operation. A CPUSHL instruction causes the selected line to remain valid.
11 IHLCK	Instruction cache half-lock. 0 Normal operation. The cache allocates to the lowest invalid way; if all ways are valid, the cache allocates to the way pointed at by the round-robin counter and then increments this counter. 1 Half cache operation. The cache allocates to the lowest invalid way of ways 2 and 3; if both of these ways are valid, the cache allocates to way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates way 3 and then increments the round-robin counter. This locks the contents of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions.
10 IDCM	Instruction default cache mode. For normal operations that do not hit in the RAMBARs or ACRs, this field defines the effective cache mode. 0 Cacheable 1 Cache-inhibited
9	Reserved, must be cleared.
8 ICINVA	Instruction cache invalidate. Invalidation occurs when this bit is set. Caches are not cleared on power-up or normal reset. 0 No invalidation is performed. 1 Initiate instruction cache invalidation. The cache controller clears all V bits sequentially. Subsequent local memory bus accesses stall until invalidation completes, at which point ICINVA is cleared automatically without software intervention. For copyback mode, use CPUSHL before setting ICINVA.
7 IDSP	Instruction default supervisor-protect. For normal operations that do not hit in the RAMBAR or ACRs, this field defines supervisor-protection. 0 Not supervisor protected 1 Supervisor protected. User operations cause a fault
6	Reserved, must be cleared.
5 EUSP	Enable USP. Enables user stack pointer. 0 USP disabled. Core uses a single stack pointer. 1 USP enabled. Core uses separate supervisor and user stack pointers.
4–0	Reserved, must be cleared.

6.3.2 Access Control Registers (ACR*n*)

The ACR*n* registers assign control attributes, such as cache mode and write protection, to specified memory regions. ACR0 and ACR1 control data attributes; ACR2 and ACR3 control instruction attributes. Registers are accessed with the MOVEC instruction with the Rc encodings in Figure 6-5.

For overlapping regions, the lower ACR number takes priority. Data transfers to and from these registers are longword transfers.

NOTE

ACR0–3 are read/write by the debug module.

BDM: 0x004 (ACR0)
0x005 (ACR1)
0x006 (ACR2)
0x007 (ACR3)

Access: MOVEC write-only
Debug 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
R	BA								ADMSK								E	S		0	0	AMM	0	0	0	CM		0	0	W[1]	0	0
W																													SP			
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	–	–	0	0	0	0	0	0	–	–	0	0	–	0	0

[1] Reserved in ACR2 and ACR3

Figure 6-5. Access Control Register Format (ACR*n*)

Table 6-4. ACR*n* Field Descriptions

Field	Description
31–24 BA	Base address. Compared with address bits A[31:24]. Eligible addresses that match are assigned the access control attributes.
23–16 ADMSK	Address mask. Setting a mask bit causes the corresponding address base bit to be ignored. The low-order mask bits can be set to define contiguous regions larger than 16 Mbytes. The mask can define multiple non-contiguous regions of memory.
15 E	Enable. Enables or disables the other ACR*n* bits. 0 Access control attributes disabled 1 Access control attributes enabled
14–13 S	Supervisor mode. Specifies whether only user or supervisor accesses are allowed in this address range or if the type of access is a don’t care. 00 Match addresses only in user mode 01 Match addresses only in supervisor mode 1x Execute cache matching on all accesses
12–11	Reserved, must be cleared.
10 AMM	Address mask mode. 0 The ACR hit function allows control of a 16 Mbytes or greater memory region. 1 The upper 8 bits of the address and ACR are compared without a mask function. Address bits [23:20] of the address and ACR are compared using ACR[19:16] as a mask, allowing control of a 1–16 Mbyte memory region.
9–7	Reserved, must be cleared.

Table 6-4. ACR*n* Field Descriptions (continued)

Field	Description
6–5 CM	Cache mode. Selects the cache mode and access precision. Precise and imprecise modes are described in Section 6.4.1.2, "Cache-Inhibited Accesses." 00 Cacheable, write-through 01 Cacheable, copyback 10 Cache-inhibited, precise 11 Cache-inhibited, imprecise
4	Reserved, must be cleared.
3 SP	Supervisor protect. 0 Indicates supervisor and user mode access allowed 1 Indicates only supervisor access is allowed to this address space and attempted user mode accesses generate an access error exception
2 W	Write protect. Selects the write privilege of the memory region. This field is reserved in the instruction attribute ACRs (ACR2–3). 0 Read and write accesses permitted 1 Write accesses not permitted
1–0	Reserved, must be cleared.

6.4 Functional Description

Figure 6-6 shows the general flow of a caching operation using the 16-Kbyte data cache as an example. This chapter assumes a data cache. Instruction cache operations are similar except for writing to the cache has no support; therefore, such notions of modified cache lines and write allocation do not apply.

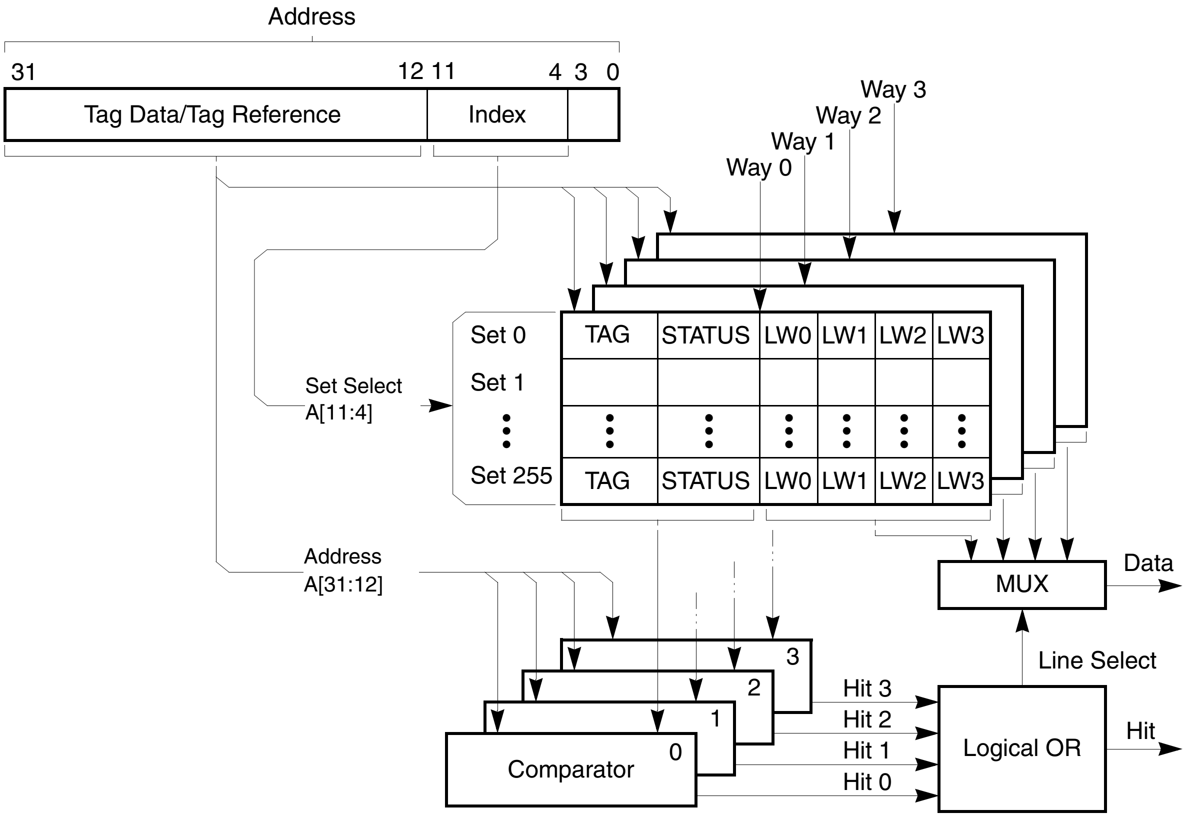


Figure 6-6. Data-Caching Operation

The following steps determine if a data-cache line for a given address is allocated:

1. The cache set index, A[11:4], selects one cache set.
2. A[31:12] and the cache set index are used as a tag reference or to update the cache line tag field. A[31:12] can specify 20 possible address lines that can be mapped to one of the four ways.
3. The four tags from the selected cache set are compared with the tag reference. A cache hit occurs if a tag matches the tag reference and the V bit is set, indicating that the cache line contains valid data. If a cacheable write access hits in a valid cache line, the write can occur to the cache line without loading it from memory.

   If the memory space is copyback, the updated cache line is marked modified (M = 1), because the new data made the data in memory stale. If the memory location is write-through, the write is passed to system memory and the M bit is not used. The tag does not have TT or TM bits.

To allocate a cache entry, the cache set index selects one of the cache's 256 sets. The cache control logic looks for an invalid cache line to use for the new entry. If none are available, the cache controller uses a pseudo-round-robin replacement algorithm to choose the line to be deallocated and replaced. First, the cache controller looks for an invalid line, with way 0 the highest priority. If all lines have valid data, a 2-bit replacement counter chooses the way. After a line is allocated, the pointer increments to point to the next way.

Cache lines from ways 0 and 1 can be protected from deallocation by enabling half-cache locking. If CACR[DHLCK,IHLCK] are set, the replacement pointer is restricted to way 2 or 3.

As part of deallocation, a valid, unmodified cache line is invalidated. It is consistent with system memory, so memory does not need to be updated. To deallocate a modified cache line, data is placed in a push buffer (for an external cache line push) before being invalidated. After invalidation, the new entry can replace it. The old cache line may be written after the new line is read.

When a cache line is selected to host a new cache entry, three things happen:

1. The new address tag bits A[31:12] are written to the tag.
2. The cache line is updated with the new memory data.
3. The cache line status changes to a valid state (V = 1).

Read cycles that miss in the cache allocate normally as previously described. Write cycles that miss in the cache do not allocate on a cacheable write-through region but do allocate for addresses in a cacheable copyback region.

A copyback byte, word, longword, or line write miss causes the following:

1. The cache initiates a line fill or flush.
2. Space is allocated for a new line.
3. V and M are set to indicate valid and modified.
4. Data is written in the allocated space. No write to memory occurs.

NOTE

Read hits cannot change the status bits and no deallocation or replacement occurs; the data or instructions are read from the cache. If the cache hits on a write access, data is written to the appropriate portion of the accessed cache line. Write hits in cacheable, write-through regions generate an external write cycle and the cache line is marked valid, but is never marked modified. Write hits in cacheable copyback regions do not perform an external write cycle; the cache line is marked valid and modified (V and M are set). Misaligned accesses are broken into at least two cache accesses. Validity is provided only on a line basis. Unless a whole line is loaded on a cache miss, the cache controller does not validate data in the cache line.

Write accesses designated as cache-inhibited by the CACR or ACR bypass the cache and perform a corresponding external write.

Normally, cache-inhibited reads bypass the cache and are performed on the external bus. The exception occurs when all of the following conditions are true during a cache-inhibited read:

- The cache-inhibited fill buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (TT = 0).

In this case, an entire line is fetched and stored in the fill buffer. It remains valid there, and the cache can service additional read accesses from this buffer until either another fill or a cache-invalidate-all operation occurs.

Valid cache entries that match during cache-inhibited address accesses are neither pushed nor invalidated. Such a scenario suggests that the associated cache mode for this address space was changed. To avoid this, use the CPUSHL instruction to push or invalidate the cache entry or set CACR[DCINVA] to invalidate the data cache before switching cache modes.

6.4.1 Caching Modes

For every memory reference the processor or debug module generates, a set of effective attributes is determined based on the address and ACRs. Caching modes determine how the cache handles an access. A data access can be cacheable in write-through or copyback mode; it can be cache-inhibited in precise or imprecise modes. For normal accesses, the ACR*n*[CM] bit corresponding to the address of the access specifies the caching modes. If an address does not match an ACR, the default caching mode is defined by CACR[DDCM,IDCM]. The specific algorithm is as follows:

```
if (address == ACR0-address including mask)
        effective attributes = ACR0 attributes
else if (address == ACR1-address including mask)
        effective attributes = ACR1 attributes
else effective attributes = CACR default attributes
```

Addresses matching an ACR can also be write-protected using ACR[W]. Addresses that do not match either ACR can be write-protected using CACR[DW].

Reset disables the cache and clears all CACR bits. As shown in Figure 6-3, reset does not automatically invalidate cache entries; the software invalidates them.

The ACRs allow the defaults selected in the CACR to be overridden. In addition, some instructions (for example, CPUSHL) and processor core operations perform accesses that have an implicit caching mode associated with them. The following sections discuss the different caching accesses and their associated cache modes.

6.4.1.1 Cacheable Accesses

If ACR*n*[CM] or the default field of the CACR indicates write-through or copyback, the access is cacheable. If matching data is found, a read access to a write-through or copyback region is read from the cache. Otherwise, the data is read from memory, and the cache is updated. When a line is read from memory for either a write-through or copyback read miss, the longword within the line that contains the core-requested data is loaded first, and the requested data is given immediately to the processor, without waiting for the three remaining longwords to reach the cache.

The following sections describe write-through and copyback modes in detail. Some of this information applies to data caches only.

6.4.1.1.1 Write-Through Mode (Data Cache Only)

Write accesses to regions specified as write-through are always passed on to the external bus; although the cycle can be buffered, depending on the state of CACR[DESB]. Writes in write-through mode are handled with a no-write-allocate policy—that is, writes that miss in the cache are written to the external bus but do not cause the corresponding line in memory to load into the cache. Write accesses that hit always write through to memory and update matching cache lines. The cache supplies data to data-read accesses that hit in the cache; read misses cause a new cache line to load into the cache.

6.4.1.1.2 Copyback Mode (Data Cache Only)

Copyback regions are typically used for local data structures or stacks to minimize external bus use and reduce write-access latency. Write accesses to regions specified as copyback that hit in the cache update the cache line and set the corresponding M bit without an external bus access.

The cache should be flushed using the CPUSHL instruction before invalidating the cache in copyback mode using the CINV bits. Modified cache data is written to memory only if the line is replaced because of a miss or a CPUSHL instruction pushes the line. If a byte, word, longword, or line write access misses in the cache, the required cache line is read from memory, thereby updating the cache. When a miss selects a modified cache line for replacement, the modified cache data moves to the push buffer. The replacement line is read into the cache, and the push buffer contents are then written to memory.

6.4.1.2 Cache-Inhibited Accesses

Memory regions can be designated as cache-inhibited, which is useful for memory containing targets such as I/O devices and shared data structures in multiprocessing systems. It is also important to not cache the processor’s memory-mapped registers. If the corresponding ACR*n*[CM] or CACR[DDCM] indicates cache-inhibited, precise or imprecise, the access is cache-inhibited. The caching operation is identical for both cache-inhibited modes, which differ only regarding recovery from an external bus error.

In determining whether a memory location is cacheable or cache-inhibited, the CPU checks memory-control registers in the following order:

1. RAMBARs
2. ACR0 and ACR2
3. ACR1 and ACR3
4. If an access does not hit in the RAMBARs or the ACRs, the default is provided for all accesses in CACR.

Cache-inhibited write accesses bypass the cache, and a corresponding external write is performed. Cache-inhibited reads bypass the cache and are performed on the external bus, except when all of the following conditions are true:

- The cache-inhibited fill buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (TT = 0).

In this case, a fetched line is stored in the fill buffer and remains valid there; the cache can service additional read accesses from this buffer until another fill occurs or a cache-invalidate-all operation occurs.

If ACR*n*[CM] indicates cache-inhibited mode, precise or imprecise, the controller bypasses the cache and performs an external transfer. If a line in the cache matches the address and the mode is cache-inhibited, the cache does not automatically push the line if it is modified, nor does it invalidate the line if it is valid. Before switching cache mode, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] to invalidate the entire cache.

If ACR*n*[CM] indicates precise mode, the sequence of read and write accesses to the region is guaranteed to match the instruction sequence. In imprecise mode, the processor core allows read accesses that hit to occur before completion of a pending write from a previous instruction. Writes are not deferred past data-read accesses that miss the cache (they must be read from the bus).

Precise operation forces data-read accesses for an instruction to occur only once by preventing the instruction from being interrupted after data is fetched. Otherwise, if the processor is not in precise mode, an exception aborts the instruction and the data may be accessed again when the instruction is restarted. These guarantees apply only when ACR*n*[CM] indicates precise mode and aligned accesses.

All CPU space-register accesses, such as MOVEC, are treated as cache-inhibited and precise.

6.4.2 Cache Protocol

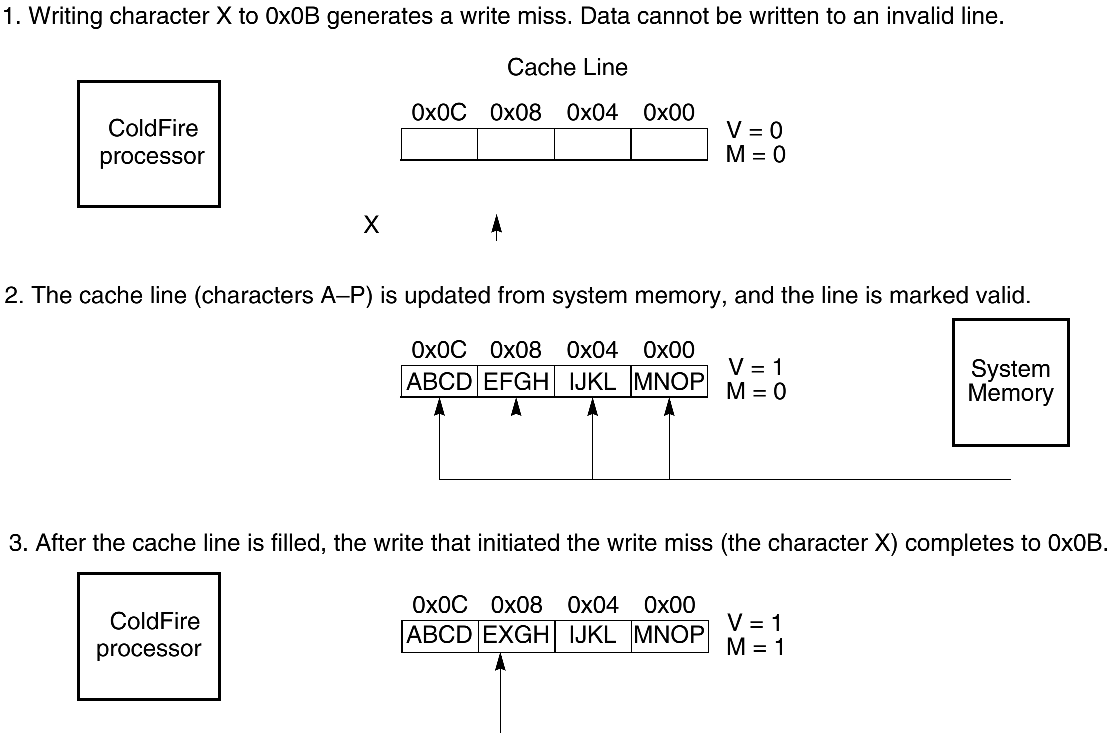
The following sections describe the cache protocol for processor accesses and assumes that the data is cacheable (that is, write-through or copyback). The discussion of write operations applies to the data cache only.

6.4.2.1 Read Miss

A processor read that misses in the cache requests the cache controller to generate a bus transaction. This bus transaction reads the needed line from memory and supplies the required data to the processor core. The line is placed in the cache in the valid state.

6.4.2.2 Write Miss (Data Cache Only)

The cache controller handles processor writes that miss in the data cache differently for write-through and copyback regions. Write misses to copyback regions cause the cache line to be read from system memory, as shown in Figure 6-7.



1. Writing character X to 0x0B generates a write miss. Data cannot be written to an invalid line.

2. The cache line (characters A–P) is updated from system memory, and the line is marked valid.

3. After the cache line is filled, the write that initiated the write miss (the character X) completes to 0x0B.

Figure 6-7. Write-Miss in Copyback Mode

The new cache line is then updated with write data and the M bit is set for the line, leaving it in modified state. Write misses to write-through regions write directly to memory without loading the corresponding cache line into the cache.

6.4.2.3 Read Hit

On a read hit, the cache provides the data to the processor core and the cache line state remains unchanged. If the cache mode changes for a specific region of address space, lines in the cache corresponding to that region that contain modified data are not pushed out to memory when a read hit occurs within that line. First, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching the cache mode.

6.4.2.4 Write Hit (Data Cache Only)

The cache controller handles processor writes that hit in the data cache differently for write-through and copyback regions. For write hits to a write-through region, portions of cache lines corresponding to the size of the access are updated with the data. The data is also written to external memory. The cache line state is unchanged. For copyback accesses, the cache controller updates the cache line and sets the M bit for the line. An external write is not performed and the cache line state changes to (or remains in) the modified state.

6.4.3 Cache Coherency (Data Cache Only)

The processor provides limited support for maintaining cache coherency in multiple-master environments. Write-through and copyback memory update techniques are supported to maintain coherency between the cache and memory.

The cache does not support snooping (cache coherency is not supported while external or DMA masters use the bus). Therefore, on-chip DMA channels should not access cached local memory locations because coherency is not maintained with the data cache.

6.4.4 Memory Accesses for Cache Maintenance

The cache controller performs all maintenance activities that supply data from the cache to the core, including requests for reading new cache lines and writing modified cache lines to memory. The following sections describe memory accesses resulting from cache fill and push operations.

6.4.4.1 Cache Filling

When a new cache line is required, a line read is requested, which generates a burst-read transfer by indicating a line access with the size signals, SIZ[1:0].

The responding device supplies four consecutive longwords of data. Line accesses implicitly request burst-mode operations from memory, but burst operations can be inhibited or enabled through the burst read/write enable bits (CSCR*n*[BSTR, BSTW]). For more information regarding external bus burst-mode accesses, see Chapter 20, “FlexBus.”

The first cycle of a cache-line read loads the longword entry corresponding to the requested address. Subsequent transfers load the remaining longword entries.

A burst operation aborts by a write-protection fault, which is the only possible access error. Exception processing proceeds immediately. Unlike Version 2 and Version 3 access errors, the program counter stored on the exception stack frame points to the faulting instruction. See Section 3.3.4.1, “Access Error Exception.”

6.4.4.2 Cache Pushes

Cache pushes occur for line replacement and as required for the execution of the CPUSHL instruction. To reduce the requested data’s latency in the new line, the modified line being replaced is temporarily placed in the push buffer while the new line is fetched from memory. After the bus transfer for the new line completes, the modified cache line writes back to memory and the push buffer invalidates.

6.4.4.2.1 Push and Store Buffers

The 16-byte push buffer reduces latency for requested new data on a cache miss by holding a displaced modified data cache line while the new data is read from memory.

If a cache miss displaces a modified line, a miss read reference is immediately generated. While waiting for the response, the current contents of the cache location load into the push buffer. When the burst-read bus transaction completes, the cache controller can generate the appropriate line-write bus transaction to write the push buffer contents into memory.

In imprecise mode, the FIFO store buffer can defer pending writes to maximize performance. The store buffer can support as many as four entries (16 bytes maximum) for this purpose.

Data writes destined for the store buffer cannot stall the core. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (one per cycle maximum) and the external bus's ability to retire those writes. In imprecise mode, writes stall only if the store buffer is full and a write operation is on the internal bus. The internal write cycle is held, stalling the data execution pipeline.

If the store buffer is not used (store buffer disabled or cache-inhibited precise mode), external bus cycles generate directly for each pipeline-write operation. The instruction is held in the pipeline until external bus transfer termination is received. Therefore, each write is stalled for five cycles, making the minimum write time equal to six cycles when the store buffer is not used. See Section 3.1.1.2, "Operand Execution Pipeline (OEP)."

The data store buffer enable bit, CACR[DESB], controls the enabling of the data-store buffer. The MOVEC instruction can set and clear this bit. At reset, this bit is cleared and all writes perform in order (precise mode). ACR*n*[CM] or CACR[DDCM] generates the mode used when DESB is set. Cacheable write-through and cache-inhibited imprecise modes use the store buffer.

The store buffer can queue data as much as four bytes wide per entry. Each entry matches the corresponding bus cycle it generates; therefore, a misaligned longword write to a write-through region creates two entries if the address is to an odd-word boundary. It creates three entries if the address is to an odd-byte boundary—one per bus cycle.

6.4.4.2.2 Push and Store Buffer Bus Operation

As soon as the push or store buffer has valid data, the internal bus controller uses the next available external bus cycle to generate the appropriate write cycles. In the event another cache fill is required (for example, cache miss to process) during the continued instruction execution by the processor pipeline, the pipeline stalls until the push and store buffers empty, before generating the required external bus transaction.

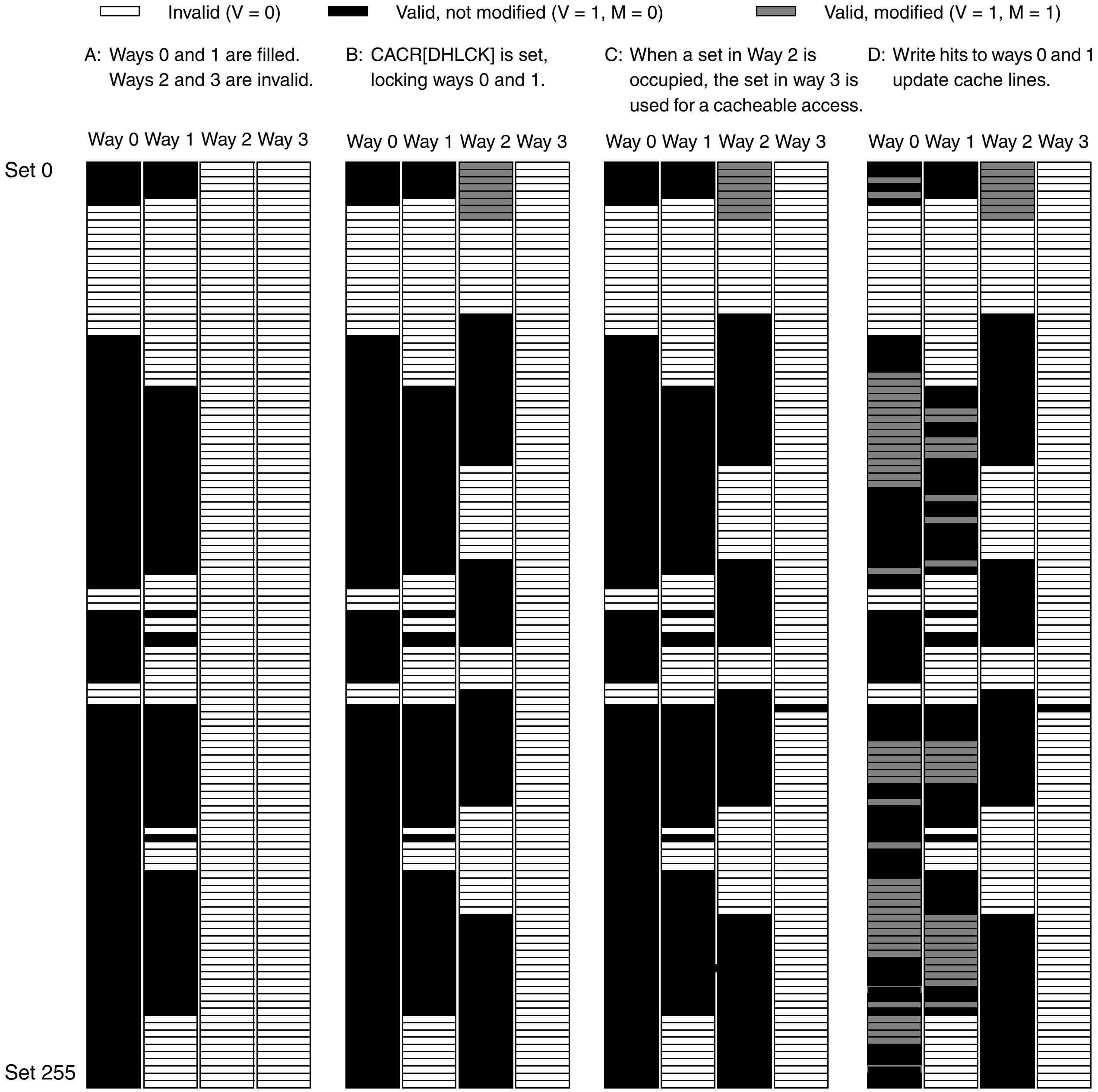
Supervisor instructions, the NOP instruction, and exception processing synchronize the processor core and guarantee the push and store buffers are empty before proceeding. The NOP instruction should be used only to synchronize the pipeline. The preferred no-op function is the TPF instruction. See the *ColdFire Programmer's Reference Manual* for more information on the TPF instruction.

6.4.5 Cache Locking

Ways 0 and 1 of the data cache can lock by setting CACR[DHLCK]; likewise, ways 0 and 1 of the instruction cache can lock by setting CACR[IHLCK]. If a cache locks, cache lines in ways 0 and 1 are not subject to deallocation by normal cache operations.

As Figure 6-8 (B and C) shows, the algorithm for updating the cache and for identifying cache lines for deallocation does not change. If ways 2 and 3 are entirely invalid, cacheable accesses are first allocated in way 2. Way 3 is not used until the location in way 2 is occupied.

Ways 0 and 1 are still updated on write hits (D in Figure 6-8) and may be pushed or cleared only explicitly by using specific cache push/invalidate instructions. However, new cache lines cannot be allocated in ways 0 and 1.


Invalid (V = 0)
Valid, not modified (V = 1, M = 0)
Valid, modified (V = 1, M = 1)
A: Ways 0 and 1 are filled. Ways 2 and 3 are invalid.
B: CACR[DHLCK] is set, locking ways 0 and 1.
C: When a set in Way 2 is occupied, the set in way 3 is used for a cacheable access.
D: Write hits to ways 0 and 1 update cache lines.
Way 0 Way 1 Way 2 Way 3
Way 0 Way 1 Way 2 Way 3
Way 0 Way 1 Way 2 Way 3
Way 0 Way 1 Way 2 Way 3
Set 0
Set 255

After reset, the cache is invalidated, ways 0 and 1 are then written with data that should not be deallocated. Ways 0 and 1 can be filled systematically by using the INTOUCH instruction.

After CACR[DHLCK] is set, subsequent cache accesses go to ways 2 and 3.

While the cache is locked and after a position in ways 2 is full, the set in Way 3 is updated.

While the cache is locked, ways 0 and 1 can be updated by write hits. In this example, memory is configured as copyback, so updated cache lines are marked modified.

Figure 6-8. Data Cache Locking

6.4.6 Cache Management

The cache can be enabled and configured by using a MOVEC instruction to access CACR. A hardware reset clears CACR, disabling the cache and removing all configuration information; however, reset does not affect the tags, state information, and data in the cache.

Set CACR[DCINVA,ICINVA] to invalidate the caches before enabling them.

The privileged CPUSHL instruction supports cache management by selectively pushing and invalidating cache lines. The address register used with CPUSHL directly addresses the cache’s directory array. The CPUSHL instruction flushes a cache line.

The value of CACR[DDPI,IDPI] determines whether CPUSHL invalidates a cache line after it is pushed. To push the entire cache, implement a software loop to index through all sets and each of the four lines within each set (for a total of 512 lines for the data cache and 1024 lines for the instruction cache). The state of CACR[DEC,IEC] does not affect the operation of CPUSHL or CACR[DCINVA,ICINVA]. Disabling a cache by clearing CACR[IEC] or CACR[DEC] makes the cache non-operational without affecting tags, state information, or contents.

The contents of A*x* used with CPUSHL specify cache row and line indexes. This differs from the MC68040 family where a physical address is specified. Figure 6-9 shows the A*x* format for the data and instruction cache.

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Set Index									Way Index			

Figure 6-9. A*x* Format

The following code example flushes the entire data cache:

```
_cache_disable:
         nop
         move.w           #0x2700,SR        ;mask off IRQs
         jsr              _cache_flush      ;flush the cache completely
         clr.l            d0
         movec            d0,ACR0           ;ACR0 off
         movec            d0,ACR1           ;ACR1 off
         move.l           #0x01000000,d0    ;Invalidate and disable cache
         movec            d0,CACR
         rts
_cache_flush:
         nop                                ;synchronize—flush store buffer
         moveq.l          #0,d0             ;initialize way counter
         moveq.l          #0,d1             ;initialize set counter
         move.l           d0,a0             ;initialize cpushl pointer
setloop:
         cpushl           dc,(a0)           ;push cache line a0
         add.l            #0x0010,a0        ;increment set index by 1
         addq.l           #1,d1             ;increment set counter
         cmpi.l           #255,d1           ;are sets for this way done?
         bne              setloop

         moveq.l          #0,d1             ;set counter to zero again
         addq.l           #1,d0             ;increment to next way
         move.l           d0,a0             ;set = 0, way = d0
```

```
        cmpi.l           #4,d0            ;flushed all the ways?
        bne              setloop
        rts
```

The following CACR loads assume: the instruction cache has been invalidated, the default instruction cache mode is cacheable, the default data cache mode is copyback.

```
dataCacheLoadAndLock:
        move.l   #0xa3080800,d0    ; enable and invalidate data cache ...
        movec    d0,cacr           ; ... in the CACR
```

The following code segments preload half of the data cache (8 Kbytes). It assumes a contiguous block of data is to be mapped into the data cache, starting at a 0-modulo-8K address.

```
        move.l   #512,d0           ; 512 16-byte lines in 8K space
        lea      data_,a0          ; load pointer defining data area
dataCacheLoop:
        tst.b    (a0)              ; touch location + load into data cache
        lea      16(a0),a0         ; increment address to next line
        subq.l   #1,d0             ; decrement loop counter
        bne.b    dataCacheLoop     ; if done, then exit, else continue

; A 8K region has been loaded into ways 0 and 1 of the 16K data cache. lock it!
        move.l   #0xaa088000,d0    ; set the data cache lock bit ...
        movec    d0,cacr           ; ... in the CACR
        rts

        align    16
```

The following CACR loads assume the data cache has been previously invalidated, the default instruction cache mode is cacheable, and the default operand cache mode is copyback.

This function must be mapped into a cache-inhibited or SRAM space or these text lines are to be prefetched into the instruction cache. This may displace some of the 8-Kbyte space being explicitly fetched.

```
instructionCacheLoadAndLock:
        move.l   #0xa2088100,d0    ; enable and invalidate the instruction
        movec    d0,cacr           ; cache in the CACR
```

The following code segments preload half of the instruction cache (8 Kbytes). It assumes a contiguous block of data is to be mapped into the cache, starting at a 0-modulo-8K address

```
        move.l   #512,d0           ; 512 16-byte lines in 8K space
        lea      code_,a0          ; load pointer defining code area
instCacheLoop:
        intouch  (a0)              ; touch location + load into instruction cache

; Note in the assembler we use, there is no INTOUCH opcode. The following
; is used to produce the required binary representation
        cpushl   #nc,(a0)          ;touch location + load into
                                   ;instruction cache
        lea      16(a0),a0         ;increment address to next line
        subq.l   #1,d0             ;decrement loop counter
        bne.b    instCacheLoop     ;if done, then exit, else continue
; A 8K region was loaded into levels 0 and 1 of the 16-Kbyte instruction cache. lock it!

        move.l   #0xa2088800,d0    ;set the instruction cache lock bit
        movec    d0,cacr            ;in the CACR
        rts
```

6.4.7 Cache Operation Summary

This section gives operational details for the cache and presents instruction and data cache-line state diagrams.

6.4.7.1 Instruction Cache State Transitions

Because the instruction cache does not support writes, it supports fewer operations than the data cache. As Figure 6-10 shows, an instruction cache line can be in one of two states: valid or invalid. Modified state is not supported. Transitions are labeled with a capital letter (indicating the previous state) and a number (indicating the specific case listed in Table 6-5). These numbers correspond to the equivalent operations on data caches as described in Section 6.4.7.2, “Data Cache-State Transitions.”

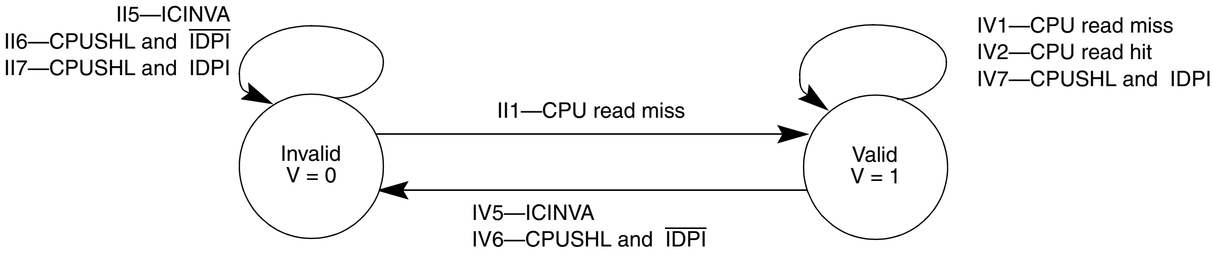


Figure 6-10. Instruction Cache Line State Diagram

Table 6-5 describes the instruction cache-state transitions shown in Figure 6-10.

Table 6-5. Instruction Cache Line State Transitions

Access	Current State			
	Invalid (V = 0)		Valid (V = 1)	
Read miss	II1	Read line from memory and update cache; supply data to processor; go to valid state.	IV1	Read new line from memory and update cache; supply data to processor; stay in valid state.
Read hit	II2	Not possible	IV2	Supply data to processor; stay in valid state.
Write miss	II3	Not possible	IV3	Not possible
Write hit	II4	Not possible	IV4	Not possible
Cache invalidate	II5	No action; stay in invalid state.	IV5	No action; go to invalid state.
Cache push	II6 II7	No action; stay in invalid state.	IV6	No action; go to invalid state.
			IV7	No action; stay in valid state.

6.4.7.2 Data Cache-State Transitions

Using the V and M bits, the data cache supports a line-based protocol allowing individual cache lines to be invalid, valid, or modified. To maintain coherency with memory, the data cache supports write-through and copyback modes, specified by the corresponding ACR[CM], or CACR[DDCM] if no ACR matches.

Read or write misses to copyback regions cause the cache controller to read a cache line from memory into the cache. If available, tag and data from memory update an invalid line in the selected set. The line state then changes from invalid to valid by setting the V bit for the line. If all lines in the row are already valid or modified, the pseudo-round-robin replacement algorithm selects one of the four lines and replaces the tag and data. Before replacement, modified lines are buffered temporarily and later copied back to memory after the new line has been read from memory.

Figure 6-11 shows the three possible data-cache line states and possible processor-initiated transitions for memory configured as copyback. Transitions are labeled with a capital letter indicating the previous state and a number indicating the specific case listed in Table 6-6.

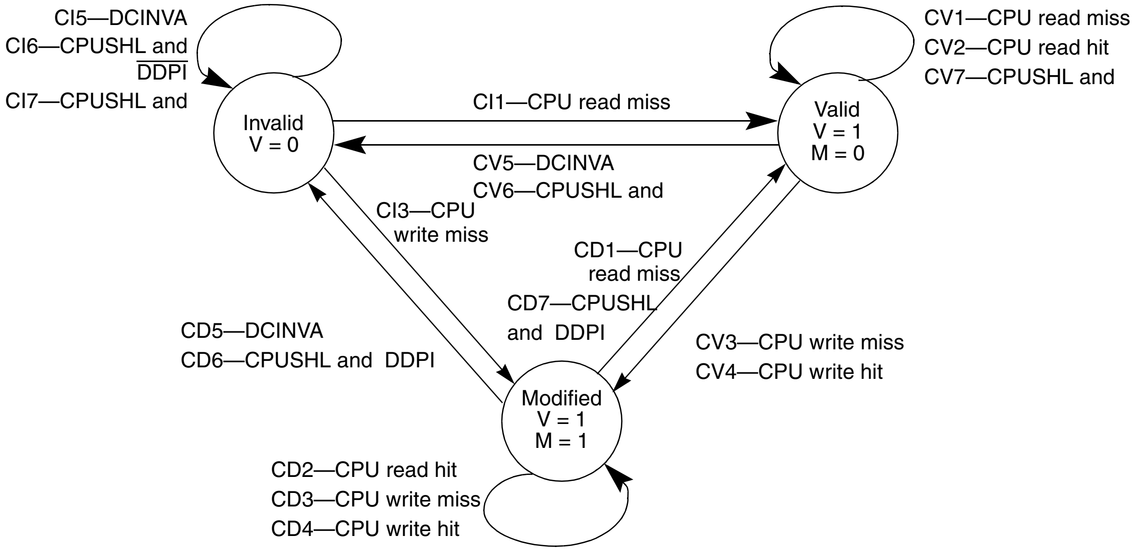


Figure 6-11. Data Cache Line State Diagram—Copyback Mode

Figure 6-12 shows the two possible states for a cache line in write-through mode.

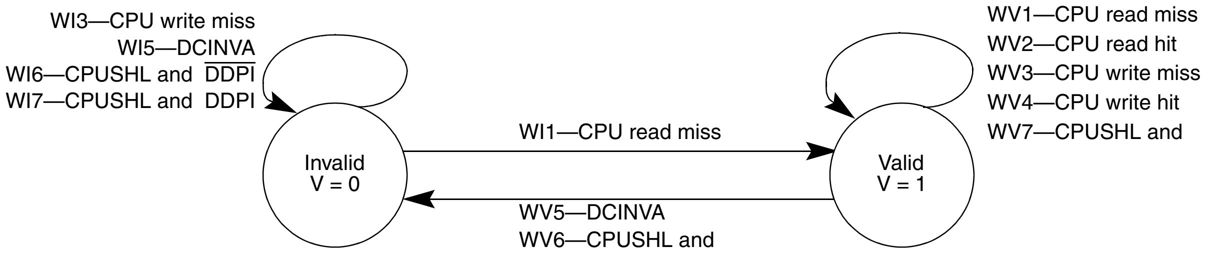


Figure 6-12. Data-Cache Line State Diagram—Write-Through Mode

Table 6-6 describes data-cache line transitions and what accesses cause them.

Table 6-6. Data Cache Line State Transitions

Access	Current State					
	Invalid (V = 0)		Valid (V = 1, M = 0)		Modified (V = 1, M = 1)	
Read miss	(C,W)I1	Read line from memory and update cache; supply data to processor; go to valid state.	(C,W)V1	Read new line from memory and update cache; supply data to processor; stay in valid state.	CD1	Push modified line to buffer; read new line from memory and update cache; supply data to processor; write push buffer contents to memory; go to valid state.
Read hit	(C,W)I2	Not possible.	(C,W)V2	Supply data to processor; stay in valid state.	CD2	Supply data to processor; stay in modified state.
Write miss (copy-back)	CI3	Read line from memory and update cache; write data to cache; go to modified state.	CV3	Read new line from memory and update cache; write data to cache; go to modified state.	CD3	Push modified line to buffer; read new line from memory and update cache; write push buffer contents to memory; stay in modified state.
Write miss (write-through)	WI3	Write data to memory; stay in invalid state.	WV3	Write data to memory; stay in valid state.	WD3	Write data to memory; stay in modified state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Write hit (copy-back)	CI4	Not possible.	CV4	Write data to cache; go to modified state.	CD4	Write data to cache; stay in modified state.
Write hit (write-through)	WI4	Not possible.	WV4	Write data to memory and to cache; stay in valid state.	WD4	Write data to memory and to cache; go to valid state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Cache invalidate	(C,W)I5	No action; stay in invalid state.	(C,W)V5	No action; go to invalid state.	CD5	No action (modified data lost); go to invalid state.
Cache push	(C,W)I6(C,W)I7	No action; stay in invalid state.	(C,W)V6	No action; go to invalid state.	CD6	Push modified line to memory; go to invalid state.
			(C,W)V7	No action; stay in valid state.	CD7	Push modified line to memory; go to valid state.

The following tables present the same information as Table 6-6, organized by the current state of the cache line. In Table 6-7 the current state is invalid.

Table 6-7. Data Cache Line State Transitions (Current State Invalid)

Access	Response	
Read miss	(C,W)I1	Read line from memory and update cache; supply data to processor; go to valid state.
Read hit	(C,W)I2	Not possible
Write miss (copyback)	CI3	Read line from memory and update cache; write data to cache; go to modified state.
Write miss (write-through)	WI3	Write data to memory; stay in invalid state.
Write hit (copyback)	CI4	Not possible
Write hit (write-through)	WI4	Not possible
Cache invalidate	(C,W)I5	No action; stay in invalid state.
Cache push	(C,W)I6	No action; stay in invalid state.
Cache push	(C,W)I7	No action; stay in invalid state.

In Table 6-8 the current state is valid.

Table 6-8. Data Cache Line State Transitions (Current State Valid)

Access	Response	
Read miss	(C,W)V1	Read new line from memory and update cache; supply data to processor; stay in valid state.
Read hit	(C,W)V2	Supply data to processor; stay in valid state.
Write miss (copyback)	CV3	Read new line from memory and update cache; write data to cache; go to modified state.
Write miss (write-through)	WV3	Write data to memory; stay in valid state.
Write hit (copyback)	CV4	Write data to cache; go to modified state.
Write hit (write-through)	WV4	Write data to memory and to cache; stay in valid state.
Cache invalidate	(C,W)V5	No action; go to invalid state.

Table 6-8. Data Cache Line State Transitions (Current State Valid) (continued)

Access	Response	
Cache push	(C,W)V6	No action; go to invalid state.
Cache push	(C,W)V7	No action; stay in valid state.

In Table 6-9 the current state is modified.

Table 6-9. Data Cache Line State Transitions (Current State Modified)

Access	Response	
Read miss	CD1	Push modified line to buffer; read new line from memory and update cache; supply data to processor; write push buffer contents to memory; go to valid state.
Read hit	CD2	Supply data to processor; stay in modified state.
Write miss (copyback)	CD3	Push modified line to buffer; read new line from memory and update cache; write push buffer contents to memory; stay in modified state.
Write miss (write-through)	WD3	Write data to memory; stay in modified state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Write hit (copyback)	CD4	Write data to cache; stay in modified state.
Write hit (write-through)	WD4	Write data to memory and to cache; go to valid state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Cache invalidate	CD5	No action (modified data lost); go to invalid state.
Cache push	CD6	Push modified line to memory; go to invalid state.
Cache push	CD7	Push modified line to memory; go to valid state.

6.4.8 CPUSHL Enhancements

The extended CPUSHL functionality adds two new bits in the cache control register (CACR) to support a set search using a physical address. In particular, the added CACR bits are defined as:

```
cacr[14] = cacr[SPA] cpushl Search by physical address
cacr[20] = cacr[IVO] cpushl Invalidate only
```

In many applications where data is shared among bus masters, the performance of the software function to push and/or clear specific lines from the data cache is important. Previously, this function was implemented using a CPUSHL loop that explicitly referenced all four ways in each cache set. By referencing all possible cache entries that might contain the targeted address range, it is guaranteed that the cache data of interest is referenced. It also has the unfortunate side-effect of potentially pushing/clearing other data that happened to be mapped into the targeted cache entries.

For the enhanced CPUSHL functionality, a higher-performance version of this function is possible using a physical address range and a simpler search loop. The enhanced CPUSHL instruction also affects only the specific cache lines being referenced and does not change the state of any other cache entries.

The specific variation of the CPUSHL instruction used to operate only on the data cache:

```
        cpushl dc, (ax)
```

where *dc* specifies the data cache, and *ax* is the cache set address and way number for the baseline CPUSHL functionality or *ax* is the physical address for the enhanced CPUSHL.

For the enhanced implementations, the specific operation performed by the CPUSHL instruction is defined by the state of four CACR bits. See Table 6-10.

Table 6-10. Enhanced CPUSHL Functionality

Instruction	CACR Bits				Description	
	[14] SPA	[20] IVO	[28] DDPI	[12] IDPI	Search by...	Action
cpushl bc,(ax)	0	0	0	0	Cache address/way	Clear both
cpushl bc,(ax)	0	0	0	1	Cache address/way	Clear data
cpushl bc,(ax)	0	0	1	0	Cache address/way	Push data, clear instruction
cpushl bc,(ax)	0	0	1	1	Cache address/way	Push data
cpushl bc,(ax)	0	1	–	–	Cache address/way	Invalidate both
cpushl bc,(ax)	1	0	0	0	Physical address	Clear both
cpushl bc,(ax)	1	0	0	1	Physical address	Clear data
cpushl bc,(ax)	1	0	1	0	Physical address	Push data, clear instruction
cpushl bc,(ax)	1	0	1	1	Physical address	Push data
cpushl bc,(ax)	1	1	–	–	Physical address	Invalidate both
cpushl dc,(ax)	0	0	0	–	Cache address/way	Clear data
cpushl dc,(ax)	0	0	1	–	Cache address/way	Push data
cpushl dc,(ax)	0	1	–	–	Cache address/way	Invalidate data
cpushl dc,(ax)	1	0	0	–	Physical address	Clear data
cpushl dc,(ax)	1	0	1	–	Physical address	Push data
cpushl dc,(ax)	1	1	–	–	Physical address	Invalidate data
cpushl ic,(ax)	0	0	–	0	Cache address/way	Clear instruction

Table 6-10. Enhanced CPUSHL Functionality (continued)

Instruction	CACR Bits				Description	
	[14] SPA	[20] IVO	[28] DDPI	[12] IDPI	Search by...	Action
`cpushl ic,(ax)`	0	0	–	1	Cache address/way	No operation
`cpushl ic,(ax)`	0	1	–	–	Cache address/way	Invalidate inst
`cpushl ic,(ax)`	1	0	–	0	Physical address	Clear instruction
`cpushl ic,(ax)`	1	0	–	1	Physical address	No operation
`cpushl ic,(ax)`	1	1	–	–	Physical address	Invalidate instruction
`cpushl nc,(ax)`	–	–	–	–	Address	`intouch` instruction

6.5 Initialization/Application Information

The following example sets up the cache for flash or ROM space only.

```
move.l   #0xA70C8100,D0           //enable cache, invalidate it,
                                  //default mode is cache-inhibited imprecise
movec    D0, CACR

move.l   #0xFF00C000,D0           //cache flash space, enable,
                                  //ignore supervisor/user, cacheable, writethrough
movec    D0,ACR0
```

Chapter 7
Static RAM (SRAM)

7.1 Introduction

This chapter describes the on-chip static RAM (SRAM) implementation, including general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.

7.1.1 Overview

The SRAM module provides a general-purpose memory block that the ColdFire processor can access in a single cycle. The location of the memory block can be specified to any 0-modulo-32K address within the 256-Mbyte address space (0x8000_0000 – 0x8FFF_FFFF). The memory is ideal for storing critical code or data structures or for use as the system stack. Because the SRAM module is physically connected to the processor's high-speed local bus, it can service processor-initiated accesses or memory-referencing commands from the debug module.

Depending on configuration information, processor references may be sent to the cache and the SRAM block simultaneously. If the reference maps into the region defined by the SRAM, the SRAM provides the data back to the processor, and the cache data is discarded. Accesses from the SRAM module are not cached.

The SRAM is dual-ported to provide access for any of the bus masters via the SRAM backdoor on the crossbar switch. The SRAM is partitioned into two physical memory arrays to allow simultaneous access to arrays by the processor core and another bus master. For more information on arbitration between multiple masters accessing the SRAM, see Chapter 15, “Crossbar Switch (XBS).”

7.1.2 Features

The major features includes:

- One 32 Kbyte SRAM
- Single-cycle access
- Physically located on the processor's high-speed local bus
- Memory location programmable on any 0-modulo-32 Kbyte address
- Byte, word, and longword address capabilities

7.2 Memory Map/Register Description

The SRAM programming model shown in Table 7-1 includes a description of the SRAM base address register (RAMBAR), SRAM initialization, and power management.

Table 7-1. SRAM Programming Model

Rc[11:0][1]	Register	Width (bits)	Access	Reset Value	Written w/ MOVEC	Section/Page
Supervisor Access Only Registers						
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	7.2.1/7-2

[1] The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 34, “Debug Module.”

7.2.1 SRAM Base Address Register (RAMBAR)

The configuration information in the SRAM base-address register (RAMBAR) controls the operation of the SRAM module.

- The RAMBAR holds the SRAM base address. The MOVEC instruction provides write-only access to this register.
- The RAMBAR can be read or written from the debug module.
- All undefined bits in the register are reserved. These bits are ignored during writes to the RAMBAR and return zeroes when read from the debug module.
- A reset clears the RAMBAR’s priority, backdoor write-protect, and valid bits, and sets the backdoor enable bit. This enables the backdoor port and invalidates the processor port to the SRAM (The RAMBAR must be initialized before the core can access the SRAM.) All other bits are unaffected.

NOTE

The only applicable address ranges for the SRAM module’s base address are 0x8000_0000 – 0x8FFF_8000. The address must be 0-modulo-32 K. Set the RAMBAR register appropriately.

By default, the RAMBAR is invalid, but the backdoor is enabled. In this state, any core accesses to the SRAM are routed through the backdoor. Therefore, the SRAM is accessible by the core, but it does not have a single-cycle access time. To ensure that the core has single-cycle access to the SRAM, set the RAMBAR[V] bit.

Any access within the memory range allocated for the on-chip SRAM (0x8000_0000-0x8FFF_FFFF) hits in the SRAM even if the address is beyond the defined size for the SRAM. This creates address aliasing for the on-chip SRAM memory. For example, writes to addresses 0x8000_0000 and 0x8000_8000 modify the same memory location. System software should ensure SRAM address pointers do not exceed the SRAM size to prevent unwanted overwriting of SRAM.

The RAMBAR contains several control fields. These fields are shown in Figure 7-1.

Rc[11:0]: 0x0C05 (RAMBAR)

Access: User write-only
Debug 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
R	BA																	0	0	0	PRIU	PRIL	BDE	WP	D/I	BWP	C/I	SC	SD	UC	UD	V
W																																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0	0	0	0	0	1	U	0	0	U	U	U	U	U	0

Figure 7-1. SRAM Base Address Register (RAMBAR)

Table 7-2. RAMBAR Field Descriptions

Field	Description
31–15 BA	Base Address. Defines the 0-modulo-32K base address of the SRAM module. By programming this field, the SRAM may be located on any 32-Kbyte boundary within the processor's 256-Mbyte address space. For proper operation, the base address must be set to between 0x8000_0000 and 0x8FFF_8000.
14–12	Reserved, must be cleared.
11–10 PRIU PRIL	Priority Bit. PRIU determines if the SRAM backdoor or CPU has priority in the upper 16K bank of memory. PRIL determines if the SRAM backdoor or CPU has priority in the lower 16K bank of memory. If a bit is set, the CPU has priority. If a bit is cleared, the SRAM backdoor has priority. Priority is determined according to the following table: **PRIU,PRIL** / **Upper Bank Priority** / **Lower Bank Priority** 00 / SRAM Backdoor / SRAM Backdoor 01 / SRAM Backdoor / CPU 10 / CPU / SRAM Backdoor 11 / CPU / CPU **Note:** The recommended setting (maximum performance) for the priority bits is 00.
9 BDE	Backdoor Enable. Allows access by non-core bus masters via the SRAM backdoor on the crossbar switch 0 Non-core crossbar switch master access to memory is disabled. 1 Non-core crossbar switch master access to memory is enabled.
8 WP	Write Protect. Allows only read accesses to the SRAM. When this bit is set, any attempted write access from the core generates an access error exception to the ColdFire processor core. 0 Allows core read and write accesses to the SRAM module 1 Allows only core read accesses to the SRAM module **Note:** This bit does not affect non-core write accesses.
7 D/I	Data/instruction bus. Determines if the SRAM is connected to the internal data or instruction bus. 0 Data bus 1 Instruction bus
6 BWP	Backdoor Write Protect. Allows only read accesses from the non-core bus masters. When this bit is set, any attempted write access from the non-core bus masters on the backdoor terminates the bus transfer with an access error. 0 Allows read and write accesses to the SRAM module from non-core masters. 1 Allows only read accesses to the SRAM module from non-core masters.

Table 7-2. RAMBAR Field Descriptions (continued)

Field	Description
5–1 C/I, SC, SD, UC, UD	Address Space Masks (AS*n*). These five bit fields allow types of accesses to be masked or inhibited from accessing the SRAM module. The address space mask bits are: C/I = CPU space/interrupt acknowledge cycle mask SC = Supervisor code address space mask SD = Supervisor data address space mask UC = User code address space mask UD = User data address space mask For each address space bit: 0 An access to the SRAM module can occur for this address space 1 Disable this address space from the SRAM module. If a reference using this address space is made, it is inhibited from accessing the SRAM module and is processed like any other non-SRAM reference. These bits do not affect accesses by non-core bus masters using the SRAM backdoor port in any manner. These bits are useful for power management as detailed in Section 7.3.2, “Power Management.” In most applications, the C/I bit is set
0 V	Valid. When set, this bit enables the SRAM module; otherwise, the module is disabled. A hardware reset clears this bit. 0 Processor accesses of the SRAM are masked 1 Processor accesses of the SRAM are enabled

7.3 Initialization/Application Information

After a hardware reset, the SRAM module contents are undefined. The valid bit of the RAMBAR is cleared, disabling the processor port into the memory. RAMBAR[BDE] is set, enabling the system backdoor port into the memory. If the SRAM requires initialization with instructions or data, perform the following steps:

1. Load the RAMBAR, mapping the SRAM module to the desired location within the address space.
2. Read the source data and write it to the SRAM. Various instructions support this function, including memory-to-memory move instructions, or the MOVEM opcode. The MOVEM instruction is optimized to generate line-sized burst fetches on 0-modulo-16 addresses, so this opcode generally provides maximum performance.
3. After the data loads into the SRAM, it may be appropriate to load a revised value into the RAMBAR with a new set of attributes. These attributes consist of the write-protect and address space mask fields.

The ColdFire processor or an external debugger using the debug module can perform these initialization functions.

7.3.1 SRAM Initialization Code

The following code segment describes how to initialize the SRAM. The code sets the base address of the SRAM at 0x8000_0000 and initializes the SRAM to zeros.

```
RAMBASE          EQU 0x80000000           ;set this variable to 0x80000000
RAMVALID         EQU 0x00000001
```

```
         move.l   #RAMBASE+RAMVALID,D0         ;load RAMBASE + valid bit into D0.
         movec.l  D0, RAMBAR                   ;load RAMBAR and enable SRAM
```

The following loop initializes the entire SRAM to zero:

```
         lea.l    RAMBASE,A0                   ;load pointer to SRAM
         move.l   #8192,D0                     ;load loop counter into D0 (SRAM size/4)

SRAM_INIT_LOOP:
         clr.l    (A0)+                        ;clear 4 bytes of SRAM
         clr.l    (A0)+                        ;clear 4 bytes of SRAM
         clr.l    (A0)+                        ;clear 4 bytes of SRAM
         clr.l    (A0)+                        ;clear 4 bytes of SRAM
         subq.l   #4,D0                        ;decrement loop counter
         bne.b    SRAM_INIT_LOOP               ;if done, then exit; else continue looping
```

7.3.2 Power Management

As noted previously, depending on the RAMBAR-defined configuration, instruction fetch and operand read accesses may be sent to the SRAM and cache simultaneously. If the access maps to the SRAM module, it sources the read data and the cache access is discarded. If the SRAM is used only for data operands, setting the AS*n* bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. Table 7-3 shows examples of typical RAMBAR settings.

Table 7-3. Typical RAMBAR Setting Examples

Data Contained in SRAM	RAMBAR[7:0]
Instruction Only	0x2B
Data Only	0x35
Instructions and Data	0x21

Chapter 8
Clock Module

8.1 Introduction

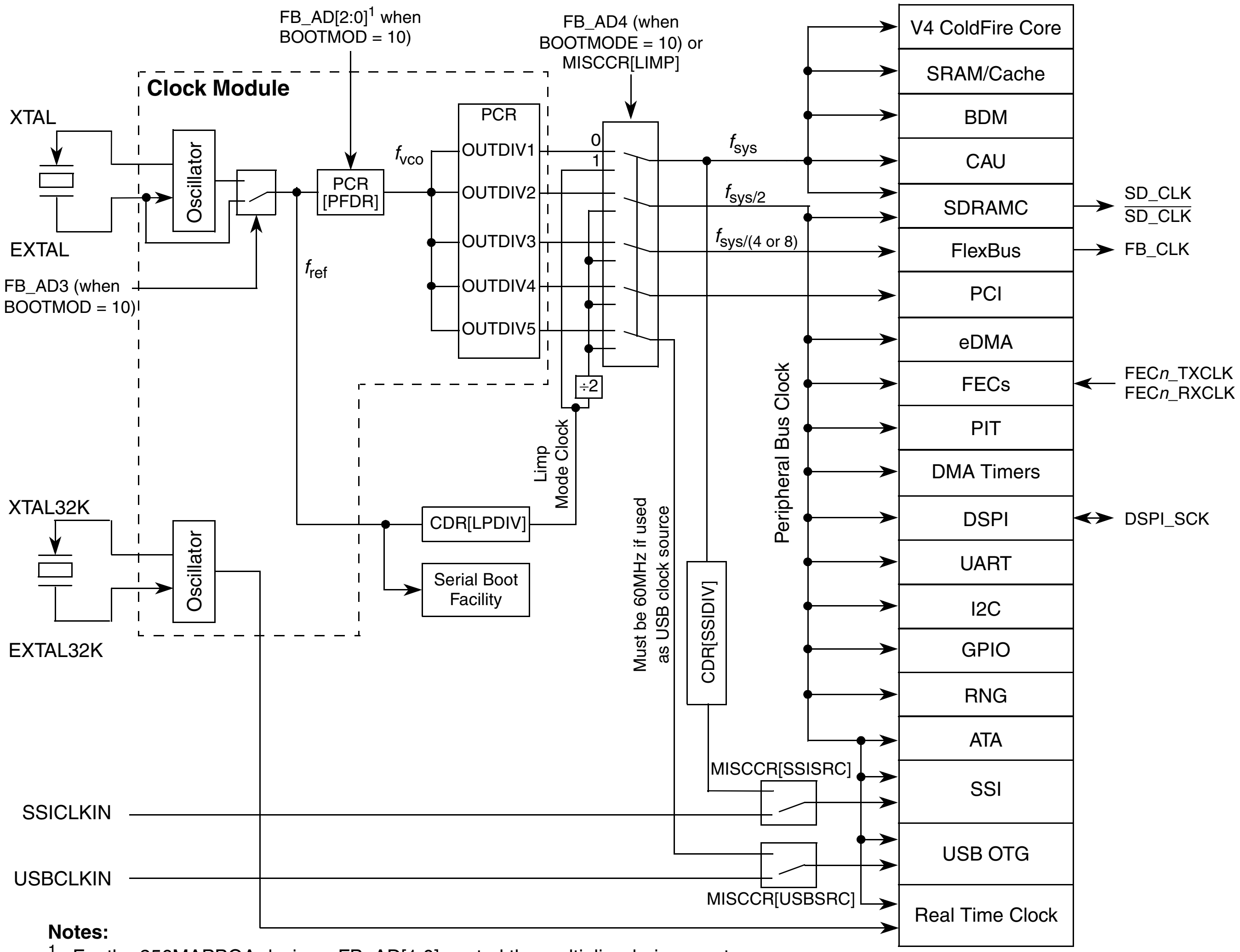
The clock module allows the device to be configured for one of several clocking methods. Clocking modes include internal phase-locked loop (PLL) clocking with an external clock reference or an external crystal reference supported by an internal crystal amplifier. The PLL can also be disabled, and an external oscillator can directly clock the device. The clock module contains:

- Crystal amplifier and oscillator (OSC)
- Phase-locked loop (PLL)
- Status and control registers
- Control logic

NOTE

Throughout this manual, f_{sys} refers to the core frequency and $f_{sys/2}$ refers to the internal bus frequency.

Figure 8-1 is a high-level representation of clock connections. The exact functionality of the blocks is not illustrated (SBF controls many configuration options, clocks to the SDRAMC, USB, FECs, PCI, and ATA controllers are disabled when the device is in limp mode, and the clocks to individual modules may be disabled via the peripheral power management registers as described in Chapter 9, “Power Management”).



Notes:

[1] For the 256MAPBGA devices, FB_AD[1:0] control the multiplier during reset.

[2] The output frequency of OUTDIV2 must equal the output frequency of OUTDIV1 ÷ 2.

[3] The output frequency of OUTDIV3 must equal the output frequency of OUTDIV1 ÷ 4 or OUTDIV ÷ 8.

[4] The output frequency of OUTDIV5 must be 60MHz if it is used as the USB clock source.

[5] The SDRAMC, FECs, and PCI modules are disabled in limp mode. The USB controller is essentially disabled, as well. However, it is able to capture a wake-up event to bring the device out of limp mode.

[6] The SDRAMC, SSI, USB, and real time clock contain some logic that uses the $f_{sys/2}$ clock, in addition to the module-specific clock.

[7] When loading boot code via the SBF, the device is clocked by the main oscillator (f_{ref}).

Figure 8-1. Device Clock Connections

8.1.1 Block Diagram

Figure 8-2 shows the clock module block diagram.

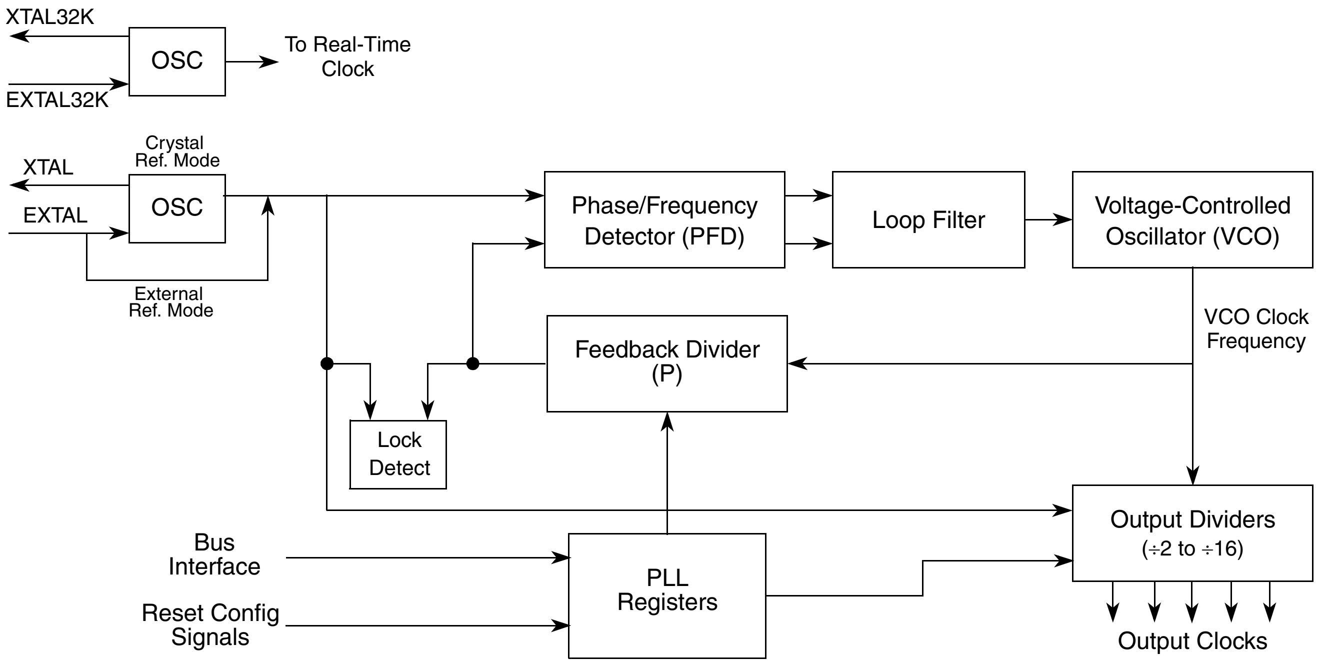


Figure 8-2. Clock Module Diagram

8.1.2 Features

Features of the clock module include:

- 16–66.66 MHz input clock frequency
- Programmable frequency multiplication factor settings generating voltage-controlled oscillator (VCO) frequencies from 300 – 540 MHz, resulting in a core frequency of 75 MHz (f_{vco} ÷ 4) to 266.67 MHz (maximum rated frequency).
- Five user-programmable output dividers
  - — Each post-VCO divider can be programmed to divide-by-2 through divide-by-16. (There are some dependencies of the divider settings. See Section 8.2.1, “PLL Control Register (PCR)”, for details.) The post-VCO dividers can be enabled asynchronously or disabled via register.
  - — Allows glitch-free, dynamic switching of the output divider
- Provides signals indicating when the PLL has acquired lock and lost lock
- 16 – 40 MHz reference crystal oscillator
- Support for low-power modes
- Direct clocking of system by input clock, bypassing the PLL
- Loss-of-lock reset
- Reference crystal oscillator for the real time clock (RTC) module. Input clock used is programmable within the RTC.

8.1.3 Modes of Operation

The PLL operational mode must be configured during reset. The reset configuration pins must be driven to the appropriate state for the desired mode from the time $\overline{\text{RSTOUT}}$ asserts until it negates. Refer to Chapter 11, "Chip Configuration Module (CCM)."

The clock module can operate in normal PLL mode with crystal reference, normal PLL mode with external reference, and input-clock limp mode.

8.1.3.1 Normal PLL Mode with Crystal Reference

In normal mode with a crystal reference, the PLL receives an input clock frequency from the crystal oscillator circuit and multiplies the frequency to create the PLL output clock. It can synthesize frequencies ranging from 4 – 34x the input frequency. The user must supply a crystal oscillator within the appropriate input frequency range, the crystal manufacturer's recommended external support circuitry, and short signal route from the device to the crystal.

8.1.3.2 Normal PLL Mode with External Reference

This second mode is the same as Section 8.1.3.1, "Normal PLL Mode with Crystal Reference," except EXTAL is driven by an external clock generator rather than a crystal oscillator. However, the input frequency range is the same as the crystal reference. To enter normal mode with external clock generator reference, the PLL configuration must be set at reset by overriding the default reset configuration. See Chapter 11, "Chip Configuration Module (CCM)," for details on setting the device for external reference (oscillator bypass mode).

8.1.3.3 Input Clock (Limp) Mode

Through parallel RCON, serial boot, or the MISCCR[LIMP] bit, the device may be placed into a low-frequency limp mode, in which the PLL is bypassed and the device runs from a factor of the input clock (EXTAL). In this mode, EXTAL feeds a 5-bit programmable counter that divides the input clock by 2^n, where n is the value of the programmable counter field, MISCCR[LPDIV]. For more information on programming the divider, see Chapter 9, "Power Management." The programmed value of the divider may be changed without glitches or otherwise negative affects to the system.

While in this mode, the PLL is placed in bypass mode to reduce overall system power consumption. A 2:1 ratio is maintained between the core and the primary bus clock, while a 1:1 ratio is maintained between FB_CLK and the internal bus clock (normally a 1:2 ratio). Because they do not function at speeds as low as the minimum input-clock frequency, the SDRAM controller, FECs, ATA controller, and PCI controller are not functional in limp mode. The USB controller is effectively disabled as well. However, it is able to capture a wake-up event to release the processor from limp mode.

When switching from limp mode to normal functional mode, you must ensure that any peripheral transactions in progress (Ethernet frame reception/transmission) are allowed to complete to avoid data loss or corruption.

Entering limp mode via the MISCCR[LIMP] bit requires a special procedure for the SDRAM module. As noted above, the SDRAM controller is disabled in limp mode, so follow these two critical steps before setting the MISCCR[LIMP] bit:

1. Code execution must be transferred to another memory resource. Primary options are whatever memory device is attached to the FlexBus boot chip-select or on-chip SRAM (but not the CPU cache, as it may have to be flushed upon limp mode entrance or exit).
2. The SDRAM controller must be placed in self-refresh mode to avoid data loss while the SDRAMC shuts down.

8.1.3.4 Low-power Mode Operation

This subsection describes the clock module operation in low-power and halted modes of operation. Low-power modes are described in Chapter 9, “Power Management.” Table 8-1 shows the clock module operation in low-power modes.

Table 8-1. Clock Module Operation in Low-power Modes

Low-power Mode	Clock Operation	Mode Exit
Wait	Clocks sent to peripheral modules only	Clock module does not cause exit, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only	Clock module does not cause exit, but normal clocking resumes upon mode exit
Stop	All system clocks disabled	Clock module does not cause exit, but clock sources are re-enabled and normal clocking resumes upon mode exit
Halted	Normal	Clock module does not cause exit

In wait and doze modes, the system clocks to the peripherals are enabled, and the clocks to the core, and SRAM are stopped. Each module can disable its clock locally at the module level.

In stop mode, all system clocks are disabled (except the real-time clock that continues to run via its external clock). There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wake-up recovery time. The PLL can be disabled in stop mode, but requires a wake-up period before it relocks. The oscillator can also be disabled during stop mode, but it requires a wake-up period to restart.

When the PLL is enabled in stop mode (LPCR[STPMD] = 00), the external FB_CLK signal can support systems using FB_CLK as the clock source. For more information about operating the PLL in stop mode, see Section 9.2.5, “Low-Power Control Register (LPCR).”

There is also a fast wake-up option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wake-up recovery time but at the risk of sending a potentially unstable clock to the system.

8.2 Memory Map/Register Definition

The PLL programming model consists of the following:

Table 8-2. PLL Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_4000	PLL Control Register (PCR)	32	R/W	See Section	8.2.1/8-6
0xFC0C_4004	PLL Status Register (PSR)	32	R/W	0x0000_0000	8.2.2/8-8

8.2.1 PLL Control Register (PCR)

The PCR register controls the feedback and output dividers for generating the core and bus clocks. For details on altering these values after reset, see Section 8.3.1, "PLL Frequency Multiplication Factor Select."

NOTE

A single longword (32-bit) write to the PCR register is required. If back-to-back word or longword writes are attempted, some of the clocks in the system change frequency before others, which can cause the device to hang.

Address: 0xFC0C_4000 (PCR)

Access: User 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
R	PFDR								0	1	1	1	OUTDIV5				OUTDIV4				OUTDIV3				OUTDIV2				OUTDIV1			
W																																
Reset 360 TEPBGA	See Note								0	1	1	1	0	1	1	1	See Note				0	1	1	1	0	0	1	1	0	0	0	1
Reset 256 MAPBGA	See Note								0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	0	1	1	0	0	0	1

Note: The reset values of PFDR and OUTDIV4 depend on the boot configuration mode.

BOOTMOD[1:0]	360 TEPBGA		256 MAPBGA	
	PFDR	OUTDIV4	PFDR	OUTDIV4
00	0x06	0x5	0x10	0x7
01 (Reserved)				
10 (Parallel Boot)	FB_AD[1:0]	If FB_AD[6:5]=11, 0x7 Else, PFDR - 1	FB_AD[2:0]	0x7
11 (Serial Boot)	SBF_RCON [119:112]	If SBF_RCON[125]=1, 0x7 If SBF_RCON[125]=0, PFDR - 1	SBF_RCON [119:112]	0x7

Figure 8-3. PLL Control Register (PCR)

Table 8-3. PCR Field Descriptions

Field	Description
31–24 PFDR	Feedback divider for setting the VCO frequency. Valid values range from 4 (0x4) to 34 (0x22). Other settings are invalid and stable operation is not guaranteed. The reset value depends on the selected chip configuration. See Chapter 11, “Chip Configuration Module (CCM),” for more information. $$f_{VCO} = f_{REF} \times PFDR \qquad \textbf{\textit{Eqn. 8-1}}$$ where f_{REF} is the PLL input frequency from the internal oscillator or EXTAL clock source (defined by the selected chip configuration).
23–20	Reserved, must be cleared.
19–16 OUTDIV5	Output divider for generating the USB clock frequency. The divider is the value of this bit field plus 1. The reset value depends on the selected chip configuration. See Chapter 11, “Chip Configuration Module (CCM),” for more information. A value of zero disables this clock. **Note:** The OUTDIV5 resulting frequency must be 60 MHz if used as the USB clock source. $$f_{USB} = \frac{f_{VCO}}{OUTDIV5 + 1} \qquad \textbf{\textit{Eqn. 8-2}}$$
15–12 OUTDIV4	Output divider for generating the PCI clock frequency. The divider is the value of this bit field plus 1. The reset value depends on the selected chip configuration. See Chapter 11, “Chip Configuration Module (CCM),” for more information. A value of zero disables this clock. $$f_{PCI} = \frac{f_{VCO}}{OUTDIV4 + 1} \qquad \textbf{\textit{Eqn. 8-3}}$$
11–8 OUTDIV3	Output divider for generating the FlexBus clock (FB_CLK) frequency. The divider is the value of this bit field plus 1. The reset value depends on the selected chip configuration. See Chapter 11, “Chip Configuration Module (CCM),” for more information. A value of zero disables this clock. **Note:** The OUTDIV3 divider value must be four or eight times the OUTDIV1 divider. For example, if OUTDIV1 equals 0001, then OUTDIV3 must equal 0111 or 1111. FB_CLK must also not exceed 66 MHz. $$f_{FB\_CLK} = \frac{f_{SYS}}{4 \text{ or } 8} = \frac{f_{VCO}}{OUTDIV3 + 1} \qquad \textbf{\textit{Eqn. 8-4}}$$

Table 8-3. PCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–4
OUTDIV2</td><td>Output divider for generating the internal bus clock frequency. The divider is the value of this bit field plus one. The reset value depends on the chip configuration selected. See Chapter 11, “Chip Configuration Module (CCM),” for more information. A value of zero disables this clock.
<b>Note:</b> The OUTDIV2 divider value must be twice the OUTDIV1 divider. For example, if OUTDIV1 equals 0001, then OUTDIV2 equals 0011.
$$f_{SYS/2} = \frac{f_{SYS}}{2} = \frac{f_{VCO}}{\mathrm{OUTDIV2}+1} \quad \textit{Eqn. 8-5}$$</td></tr>
<tr><td>3–0
OUTDIV1</td><td>Output divider for generating the CPU clock frequency. The divider is the value of this bit field plus 1. The reset value depends on the selected chip configuration. See Chapter 11, “Chip Configuration Module (CCM),” for more information. A value of zero disables this clock.
$$f_{SYS} = \frac{f_{VCO}}{\mathrm{OUTDIV1}+1} \quad \textit{Eqn. 8-6}$$
<b>Note:</b> The maximum value restrictions on this field depend on the setting of OUTDIV2 and OUTDIV3, as shown below:
<table>
<tr><th>OUTDIV3
(FB_CLK)</th><th>OUTDIV2
(Internal
Peripheral Clock)</th><th>OUTDIV1
Maximum</th></tr>
<tr><td>8 × OUTDIV1 + 7
(CPU freq ÷ 8)</td><td>—</td><td>0001</td></tr>
<tr><td>4 × OUTDIV1 + 3
(CPU freq ÷ 4)</td><td>—</td><td>0011</td></tr>
<tr><td>0
(Disabled)</td><td>≠ 0
(Enabled)</td><td>0111</td></tr>
<tr><td>0
(Disabled)</td><td>0
(Disabled)</td><td>1111</td></tr>
</table>
</td></tr>
</table>

8.2.2 PLL Status Register (PSR)

The PSR register enables loss-of-lock reset and interrupt, and also indicates the PLL lock status.

Address: 0xFC0C_4004 (PSR)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	LOL RE	LOL IRQ	LOCK	LOCKS
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-4. PLL Status Register (PSR)

Table 8-4. PSR Field Descriptions

Field	Description
31–4	Reserved, must be cleared.
3 LOLRE	PLL loss of lock reset enable. Because reset clears the PSR register, if this bit is set and a loss-of-lock occurs, the user must read the reset status register (RSR) to determine a loss-of-lock condition occurred. See Chapter 13, "Reset Controller Module," for more details on RSR. 0 Loss of lock does not generate a reset. 1 Loss of lock generates a reset to the device.
2 LOLIRQ	PLL loss-of-lock interrupt enable. Enables an interrupt request to generate when the PLL loses lock. 0 Loss-of-lock does not generate an interrupt request. 1 Loss-of-lock generates an interrupt request.
1 LOCK	PLL lock status. Indicates a locked PLL. See Section 8.3.2, "Lock Conditions," for more details. 0 PLL is not locked. 1 PLL is locked.
0 LOCKS	PLL lost lock. Indicates that the PLL has lost lock. If the PFDR field changes or if an unexpected loss-of-lock condition occurs, this bit is set. This bit is sticky and the user must clear it before the PLL can write the register again. 0 PLL has not lost lock. 1 PLL has lost lock.

8.3 Functional Description

This subsection provides a functional description of the clock module.

8.3.1 PLL Frequency Multiplication Factor Select

The frequency multiplication factor of the PLL is defined by the feedback divider and output dividers. An example equation for the core frequency is given below:

$$f_{SYS} = f_{REF} \times \left(\frac{\mathrm{PCR[PFDR]}}{\mathrm{PCR[OUTDIV1]} + 1}\right) \qquad \textbf{\textit{Eqn. 8-7}}$$

where f_{sys} is the clock frequency of the ColdFire core and f_{REF} is the PLL clock source as shown in Figure 8-1. The allowable range of values for the PFDR is 4 to 34 and OUTDIV*n* is 1 to 15. However, PFDR must also be selected such that the VCO frequency (f_{REF} × PCR[PFDR]) is of the range 300 – 540 MHz. The other clocks on the processor are configurable in a similar fashion. However, there are various dependencies. See Section 8.2.1, "PLL Control Register (PCR)," for details.

The PCR[OUTDIV*n*] fields can be changed during normal operation or when the device is in limp mode. However, PCR[PFDR] can only be altered during limp mode. After a new value is written to the PCR, the PLL synchronizes the new value of the PCR with the VCO clock domain. Then, the transition from the old divider value to the new divider value takes place, such that the PLL output clocks remain glitch free. During the adjustment to the new divider value, a PLL output clock may experience an intermediate transition while the divider values are being synchronized. Following the transition period, all output clocks begin toggling at the new divider values simultaneously. The transition from the old divider value to the new divider value takes no more than 100 ns. Because the output divider transition takes a period of time to change, the PCR may not be written back-to-back without waiting 100 ns between writes.

8.3.2 Lock Conditions

The lock-detect logic monitors the reference frequency and the PLL feedback frequency to determine when frequency lock has been achieved. Phase lock is inferred by the frequency relationship, but is not guaranteed. The PLL lock status reflects in the PSR[LOCK] status bit. The lock-detect function uses two counters clocked by the reference and PLL feedback, respectively. When the reference counter has counted N cycles, the feedback counter is compared. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then, if the two counters counts continue to match, the lock criteria relaxes by one count, and the system is notified that the PLL has achieved frequency lock by setting the PSR[LOCK] bit.

After detection of lock, the lock circuitry continues monitoring the reference and feedback frequencies using the alternate count and compare process. If the counters do not match at any comparison time, then the PSR[LOCK] and PSR[LOCKS] status bits are cleared to indicate the PLL has lost lock. At this point, the lock criteria tightens and the lock detect process repeats. The alternate count sequences prevent false lock detects due to frequency aliasing while the PLL tries to lock. Alternating between a tight and relaxed lock criteria prevents the lock detect function from randomly toggling between locked and not locked status due to phase sensitivities.

In PLL bypass mode, the PSR[LOCK] bit is set 16 clock cycles after reset as described above. In this case, the signal does not indicate the PLL has locked to the input reference, but the bypass clock is present on the output. In bypass mode, no PLL lock exists.

8.3.3 Loss-of-Lock

When the PLL loses lock the PSR[LOCKS] status bit is set. If the PFDR is changed, or if an unexpected loss of lock condition occurs, the LOCKS status bit is set. While the PLL is in an unlocked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to relock. Therefore, during the re-locking process, the system-clock frequency is not well defined and may exceed the maximum system frequency, violating the system clock timing specifications. Due to this condition, using the loss-of-lock reset functionality as described in Section 8.3.3.1, “Loss of Lock Reset Request,” is recommended. After the PLL has re-locked, the PLL does not update the PSR[LOCKS] status bit. The LOCKS status bit is sticky, and the user must clear it before the PLL can write the register again.

8.3.3.1 Loss of Lock Reset Request

The PLL provides the ability to assert reset when a loss-of-lock condition occurs by programming the PSR[LOLRE] bit. Because the PSR[LOCK, LOCKS] bits are cleared after reset, the reset status register (RSR) must be read to determine a loss of lock condition occurred. See Chapter 13, “Reset Controller Module,” for more information on the RSR register. To exit reset in PLL mode, the reference must be present and the PLL must acquire lock. In PLL bypass mode, the PLL cannot lock; therefore, a loss of lock condition cannot occur, and LOLRE has no affect.

8.3.3.2 Loss of Lock Interrupt Request

By programming the PSR[LOLIRQ] bit, the PLL provides the ability to request an interrupt when a loss-of-lock condition occurs. This bit is sticky, and remains asserted until the user clears the

PSR[LOCKS] status bit. LOLIRQ provides information to the lock detect logic to let it know if an interrupt should be generated upon loss-of-lock. In PLL bypass mode, the PLL cannot lock; therefore, a loss-of-lock condition cannot occur, and the LOLIRQ has no affect.

8.3.4 System Clock Modes

The system clock source is determined during reset. By default the PLL is placed in crystal-reference mode and generates a core frequency of 10 times the input clock (internal bus 5x, FlexBus and USB clock 2.5x). The BOOTMOD pins can override the default mode. See Chapter 11, "Chip Configuration Module (CCM)," for more information on default configuration, as well as overwriting these defaults during reset.

Table 8-5 shows some of the various clocking scenarios offered on the processor. When the PCI controller is enabled, the input reference clock bypasses the oscillator and become the PCI reference clock.

NOTE

If PCI is enabled, the input reference clock must be a bypass clock (external oscillator) and must also equal the PCI operating frequency.

The PCI controller can operate at frequencies other than what is shown in Table 8-5, but ensure that the input bypass clock frequency is the PCI operating frequency. USB_CLKIN in the USB OTG column indicates that the USB On-the-Go module receives its clock from the USB_CLKIN signal rather than the PLL output.

Table 8-5. MCF54455 Clocking Scenarios (MHz)

ColdFire Core	SDRAMC (Core ÷ 2)	FlexBus (Core ÷ 4)	PCI	USB OTG	Input Reference Clock	Crystal Frequency
266.67	133.33	66.67	66.67	USB_CLKIN	66.67	—[1]
266.67	133.33	66.67	33.33	USB_CLKIN	33.33	—[1]
266.67	133.33	66.67	Disabled	USB_CLKIN	33.33, 66.67	33.33
240	120	60	60	60	60	—[1]
240	120	60	30	60	30	—[1]
240	120	60	Disabled	60	30, 60	20, 24, 30
200	100	50	66.67	USB_CLKIN	66.67	—[1]
200	100	50	33.33	USB_CLKIN	33.33	—[1]
200	100	50	Disabled	USB_CLKIN	33.33, 66.67	20, 25, 33.33
180	90	45	60	60	60	—[1]
180	90	45	30	60	30	—[1]
180	90	45	Disabled	60	30, 60	20, 24, 30

[1] When the PCI is enabled, use of a crystal oscillator is not supported.

8.3.5 Clock Operation During Reset

This section describes the PLL reset operation. Power-on reset and normal reset are described.

8.3.5.1 Power-On Reset (POR)

After VDD_PLL and the input clock are within specification, the PLL is held in reset for at least ten input clock cycles to initialize the PLL. The reset configuration signals are used to select the multiply factor of the PLL and the reset state of the PLL registers. While in reset, the PLL input clock is output to the device. After $\overline{\text{RESET}}$ de-asserts, PLL output clocks generate; however, until the PSR[LOCK] bit is set, the PLL output clock frequencies are not stable and within specification. When this bit is set, the PLL is in frequency lock.

8.3.5.2 External Reset

When $\overline{\text{RESET}}$ asserts, the PLL input clock outputs to the device, and the PLL does not begin acquiring lock until $\overline{\text{RESET}}$ is negated. The PSR[LOCK] bit is cleared and remains cleared while the PLL is acquiring lock.

CAUTION

When running in an unlocked state, the clocks the PLL generate are not guaranteed to be stable and may exceed the maximum specified frequency.

Chapter 9
Power Management

9.1 Introduction

This chapter explains the low-power operation of the device.

9.1.1 Features

These features support low-power operation:

- Four operation modes: run, wait, doze, and stop
- Ability to shut down most peripherals independently
- Ability to shut down clocks to most peripherals independently
- Ability to run the device in low-frequency limp mode
- Ability to shut down the external FB_CLK pin

9.2 Memory Map/Register Definition

The power management programming model consists of registers from the SCM and CCM memory space:

Table 9-1. Power Management Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers[1]					
0xFC04_0013	Wakeup Control Register (WCR)	8	R/W	0x00	9.2.1/9-2
0xFC04_002C	Peripheral Power Management Set Register 0 (PPMSR0)	8	W	0x00	9.2.2/9-3
0xFC04_002D	Peripheral Power Management Clear Register 0 (PPMCR0)	8	W	0x00	9.2.3/9-4
0xFC04_0030	Peripheral Power Management High Register 0 (PPMHR0)	32	R/W	0xFFFC_00D0	9.2.4/9-4
0xFC04_0034	Peripheral Power Management Low Register 0 (PPMLR0)	32	R/W	0x0000_0300	9.2.4/9-4
0xFC0A_0007	Low-Power Control Register (LPCR)	8	R/W	0x00	9.2.5/9-7
0xFC0A_0010	Miscellaneous Control Register (MISCCR)[2]	16	R/W	See Section	11.3.4/11-8
0xFC0A_0012	Clock Divider Register (CDR)[2]	16	R/W	0x0001	11.3.5/11-11

[1] User access to supervisor only address locations have no effect and result in a bus error

[2] The MISCCR and CDR registers are described in Chapter 11, “Chip Configuration Module (CCM).”

9.2.1 Wake-up Control Register (WCR)

Implementation of low-power stop mode and exit from a low-power mode via an interrupt requires communication between the core and logic associated with the interrupt controller. The WCR enables entry into low-power modes and includes the interrupt level setting needed to exit a low-power mode.

NOTE

> The setting of the low-power mode select field, WCR[LPMD], determines which low-power mode the device enters when a STOP instruction is issued.

Sequence of operations generally needed to enable this functionality:

1. The WCR register is programmed, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. After the processor stops execution, it asserts a specific processor status (PST) encoding. Issuing the STOP instruction when the WCR[ENBWCR] is set causes the SCM to enter the mode specified in WCR[LPMD].
3. The low power mode control logic processes the entry into a low power mode, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. After an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.
6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

Address: 0xFC04_0013 (WCR)　　Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	ENBWCR	0	LPMD		0	PRILVL		
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-1. Wake-up Control Register (WCR)

Table 9-2. WCR Field Descriptions

Field	Description
7 ENBWCR	Enable low-power mode entry. The mode entered is specified in WCR[LPMD]. 0 Low-power mode entry is disabled 1 Low-power mode entry is enabled.
6	Reserved, must be cleared.

Table 9-2. WCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5–4
LPMD</td><td>Low-power mode select. Used to select the low-power mode the chip enters after the ColdFire core executes the STOP instruction. To take effect, write these bits prior to instruction execution. The LPMD bits are readable and writable in all modes.
00 Run
01 Doze
10 Wait
11 Stop
<b>Note:</b> If WCR[LPMD] is cleared, the device stops executing code upon a STOP instruction. However, no clocks disable.</td></tr>
<tr><td>3</td><td>Reserved, must be cleared.</td></tr>
<tr><td>2–0
PRILVL</td><td>Exit low-power mode interrupt priority level. This field defines the interrupt priority level to exit the low-power mode:
<table>
<tr><th>PRILVL</th><th>Interrupt Level Needed to Exit Low-Power Mode</th></tr>
<tr><td>000</td><td>Any interrupt request exits low-power mode</td></tr>
<tr><td>001</td><td>Interrupt request levels [2-7] exit low-power mode</td></tr>
<tr><td>010</td><td>Interrupt request levels [3-7] exit low-power mode</td></tr>
<tr><td>011</td><td>Interrupt request levels [4-7] exit low-power mode</td></tr>
<tr><td>100</td><td>Interrupt request levels [5-7] exit low-power mode</td></tr>
<tr><td>101</td><td>Interrupt request levels [6-7] exit low-power mode</td></tr>
<tr><td>11<i>x</i></td><td>Interrupt request level [7] exits low-power mode</td></tr>
</table>
</td></tr>
</table>

9.2.2 Peripheral Power Management Set Register (PPMSR0)

The PPMSR register provides a simple mechanism to set a given bit in the PPM{H,L}R registers to disable the clock for a given peripheral module without needing to perform a read-modify write on the PPMR. The data value on a register write causes the corresponding bit in the PPM{H,L}R to be set. The SAMCD bit provides a global set function forcing the entire contents of the PPMR to set, disabling all peripheral module clocks. Reads of these registers return all zeroes.

Address: 0xFC04_002C (PPMSR0) Access: Supervisor Write-only

<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>W</td><td></td><td>SAMCD</td><td colspan="6">SMCD</td></tr>
<tr><td>Reset:</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 9-2. Peripheral Power Management Set Register (PPMSR0)

Table 9-3. PPMSR0 Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SAMCD	Set all module clock disables. 0 Set only those bits specified in the SMCD field 1 Set all bits in PPMRH and PPMRL, disabling all peripheral clocks
5–0 SMCD	Set module clock disable. Set the corresponding bit in PPM{H,L}R, disabling the peripheral clock.

9.2.3 Peripheral Power Management Clear Register (PPMCR0)

The PPMCR register provides a simple mechanism to clear a given bit in the PPMHR and PPMLR registers, enabling the clock for a given peripheral module without needing to perform a read-modify write on the PPMR. The data value on a register write causes the corresponding bit in the PPM{H,L}R to be clear. A value of 64 to 127 (setting the CAMCD bit) provides a global clear function, forcing the entire PPMR contents to clear, enabling all peripheral module clocks. Reads of these registers return all zeroes.

Address: 0xFC04_002D (PPMCR0)

Access: Supervisor Write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAMCD	CMCD					
Reset:	0	0	0	0	0	0	0	0

Figure 9-3. Peripheral Power Management Clear Register (PPMCR0)

Table 9-4. PPMCR0 Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAMCD	Clear all module clock disables. 0 Clear only those bits specified in the CMCD field 1 Clear all bits in PPMRH and PPMRL, enabling all peripheral clocks
5–0 CMCD	Clear module clock disable. Clear the corresponding bit in PPMR{H,L}, enabling the peripheral clock.

9.2.4 Peripheral Power Management Registers (PPMHR0 and PPMLR0)

The PPMR registers provide a bit map for controlling the generation of the peripheral clocks for each decoded address space. Recall each peripheral module is mapped into 16 kByte slots within the memory map. The PPMR registers provide a unique control bit for each address space that defines whether the module clock for the given space is enabled or disabled.

Because the operation of the crossbar switch and the system control module (SCM) are fundamental to the operation of the device, the clocks for these modules cannot be disabled.

Using a read-modify-write to this register directly or indirectly through writes to the PPMSR and PPMCR registers to set/clear individual bits can modify the PPMR individual bits.

Address: 0xFC04_0030 (PPMHR0) Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	CD49	CD48
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CD47	CD46	CD45	CD44	CD43	CD42	CD41	CD40	1	1	CD37	1	CD35	CD34	CD33	CD32
W																
Reset	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0

Figure 9-4. Peripheral Power Management High Register (PPMHR0)

Table 9-5. PPMHR0[CD*n*] Assignments

Slot Number	CD*n*	Peripheral
32	CD32	PIT 0
33	CD33	PIT 1
34	CD34	PIT 2
35	CD35	PIT 3
37	CD37	Edge Port
40	CD40	CCM, Reset Controller, Power Management
41	CD41	Pin Multiplexing and Control (GPIO)
42	CD42	PCI Controller
43	CD43	PCI Arbiter
44	CD44	USB On-the-Go
45	CD45	RNG
46	CD46	SDRAM Controller
47	CD47	SSI
48	CD48	ATA Controller
49	CD49	PLL

Address: 0xFC04_0034 (PPMLR0) Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CD31	CD30	CD29	CD28	0	CD26	CD25	CD24	CD23	CD22	CD21	0	CD19	CD18	CD17	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CD15	0	CD13	CD12	0	0	0	0	0	0	0	0	0	CD2	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-5. Peripheral Power Management Low Registers (PPMLR0)

Table 9-6. PPMLR0[CD*n*] Assignments

Slot Number	CD*n*	Peripheral
2	CD2	FlexBus
12	CD12	FEC0
13	CD13	FEC1
15	CD15	Real-Time Clock
17	CD17	eDMA Controller
18	CD18	Interrupt Controller 0
19	CD19	Interrupt Controller 1
21	CD21	IACK
22	CD22	I^{2}C
23	CD23	DSPI
24	CD24	UART0
25	CD25	UART1
26	CD26	UART2
28	CD28	DMA Timer 0
29	CD29	DMA Timer 1
30	CD30	DMA Timer 2
31	CD31	DMA Timer 3

Table 9-7. PPMHR and PPMLR Field Descriptions

Field	Description
CD*n*	Module slot *n* clock disable. 0 The clock for this module is enabled. 1 The clock for this module is disabled.

CAUTION

Take extreme caution when setting PPMR[CD40] to disable clocking of the CCM, reset controller, and power management modules. This may disable logic to reset the chip and disable the external bus monitor or other logic contained within these blocks.

9.2.5 Low-Power Control Register (LPCR)

The LPCR register controls chip operation and module operation during low-power modes.

Address: 0xFC0A_0007 (LPCR)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	0	FWKUP	STPMD		0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 9-6. Low-Power Control Register (LPCR)

Table 9-8. LPCR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 FWKUP	Fast wake-up. Determines whether the system clocks are enabled upon wake-up from stop mode. This bit must be written before execution of the STOP instruction for it to take effect. 0 System clocks enabled only when PLL is locked or operating normally. 1 System clocks enabled upon wake-up from stop mode, regardless of PLL lock status. **Note:** Setting this bit is potentially dangerous and unreliable. The system may behave unpredictably when using an unlocked clock, because the clock frequency could overshoot the maximum frequency of the device. **Note:** If FWKUP is set before entering stop mode, it should not be cleared upon wake-up from stop mode until after the PLL has actually acquired lock. Lock status may be obtained by reading PLL status register. Because the PLL never locks in limp mode, the FWKUP is ineffective. The system clocks are always enabled upon wake-up from stop mode, regardless of the value of FWKUP.
4–3 STPMD	FB_CLK stop mode bits. Controls the operation of the clocks, PLL, and oscillator in stop mode: STPMD / System Clocks / FB_CLK / PLL / Oscillator 00 / Disabled / Enabled / Enabled / Enabled 01 / Disabled / Disabled / Enabled / Enabled 10 / Disabled / Disabled / Disabled / Enabled 11 / Disabled / Disabled / Disabled / Disabled
2–0	Reserved, must be cleared.

STPMD	System Clocks	FB_CLK	PLL	Oscillator
00	Disabled	Enabled	Enabled	Enabled
01	Disabled	Disabled	Enabled	Enabled
10	Disabled	Disabled	Disabled	Enabled
11	Disabled	Disabled	Disabled	Disabled

9.3 Functional Description

This section discusses the functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes.

9.3.1 Peripheral Shut Down

All peripherals, except for the SCM and crossbar switch, may have the software remove their input clocks individually to reduce power consumption. See Section 9.2.4, "Peripheral Power Management Registers (PPMHR0 and PPMLR0)," for more information. A peripheral may be disabled at any time and remains disabled during any low-power mode of operation.

9.3.2 Limp mode

The device may also boot into a low-frequency limp mode, in which the PLL is bypassed and the device runs from a factor of the input clock (EXTAL). In this mode, EXTAL feeds a counter that divides the input clock by 2^n, where n is the value of the programmable counter field, CDR[LPDIV]. The programmed value of the divider may be changed without glitches or otherwise negative affects to the system. While in this mode, the PLL is placed in bypass mode to reduce overall system-power consumption.

Limp mode may be entered and exited by writing to MISCCR[LIMP].

While in this mode, a 2:1 ratio maintains between the core and the primary bus clock. Because they do not function at speeds as low as the minimum input clock frequency, the SDRAM controller, USB On-to-Go, FECs, PCI controller, and ATA controller are not functional in limp mode.

9.3.3 Low-Power Modes

The system enters a low-power mode by executing a STOP instruction. The low-power mode the device actually enters (stop, wait, or doze) depends on the setting of the WCR[LPMD] bits. Entry into any of these modes idles the CPU with no cycles active, powers down the system, and stops all internal clocks appropriately. During stop mode, the system clock is stopped low.

A wake-up event is required to exit a low-power mode and return to run mode. Wake-up events consist of any of these conditions:

- Any type of reset
- Any valid, enabled interrupt request

Exiting from low-power mode via an interrupt request requires:

- An interrupt request whose priority is higher than the value programmed in the WCR[PRILVL].
- An interrupt request whose priority is higher than the value programmed in the interrupt priority mask (I) field of the core's status register.
- An interrupt request from a source not masked in the interrupt controller's interrupt mask register.
- An interrupt request which has been enabled at the module of the interrupt's origin.

9.3.3.1 Run Mode

Run mode is the normal system operating mode. Current consumption in this mode is related directly to the system clock frequency.

9.3.3.2 Wait Mode

Wait mode is intended to stop only the CPU and memory clocks until a wake-up event is detected. In this mode, peripherals may be programmed to continue operating and can generate interrupts, causing the CPU to exit from wait mode.

9.3.3.3 Doze Mode

Doze mode affects the processor in the same manner as wait mode, except that some peripherals define individual operational characteristics in doze mode. Peripherals continuing to run and having the capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Stopped peripherals restart operation on exit from doze mode, as defined for each peripheral.

9.3.3.4 Stop Mode

Stop mode affects the processor the same as the wait and doze modes, except that all clocks to the system are stopped and the peripherals cease operation.

Stop mode must be entered in a controlled manner to ensure that any current operation is properly terminated. When exiting stop mode, most peripherals retain their pre-stop status and resume operation.

NOTE

Entering stop mode disables the SDRAMC, including the refresh counter. If SDRAM is used, code is required to ensure proper entry and exit from stop mode. See Chapter 21, “SDRAM Controller (SDRAMC),” for more information.

9.3.4 Peripheral Behavior in Low-Power Modes

The following subsections specify the operation of each module while in and when exiting low-power modes.

9.3.4.1 ColdFire Core

The ColdFire core disables during any low-power mode. No recovery time is required when exiting any low-power mode.

9.3.4.2 Internal SRAM

The SRAM is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

9.3.4.3 Clock Module

In wait and doze modes, the clocks to the CPU and SRAM stops and the system clocks to the peripherals enable. Each module may disable the module clocks locally at the module level, or the module clocks may be individually disabled by the PPMR registers (refer to Section 9.2.4, “Peripheral Power Management Registers (PPMHR0 and PPMLR0)”). In stop mode, all clocks to the system stop.

There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wake-up recovery time. The PLL can be disabled in stop mode, but requires a wake-up period before it can relock. The oscillator can also be disabled during stop mode, but requires a wake-up period to restart.

When the PLL is enabled in stop mode (LPCR[STPMD] = 00), the external FB_CLK signal can support systems using FB_CLK as the clock source. See Section 9.2.5, "Low-Power Control Register (LPCR)," for more information about operating the PLL in stop mode.

There is also a fast wake-up option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wake-up recovery time but at the risk of sending a potentially unstable clock to the system. This is also explained in Section 9.2.5, "Low-Power Control Register (LPCR)."

9.3.4.4 Chip Configuration Module

The chip configuration module is unaffected by entry into a low-power mode. If a reset exits low-power mode, chip configuration may execute if configured to do so.

9.3.4.5 Reset Controller

A power-on reset (POR) always causes a chip to reset and exit from any low-power mode.

In wait and doze modes, asserting the external $\overline{\text{RESET}}$ pin for at least four clocks causes an external reset that resets the chip and exits any low-power modes.

In stop mode, the $\overline{\text{RESET}}$ pin synchronization disables and asserting the external $\overline{\text{RESET}}$ pin asynchronously generates an internal reset and exit any low-power modes. Registers lose current values and must be reconfigured from reset state if needed.

If the core watchdog timer is still enabled during wait or doze modes, a watchdog timer timeout may generate a reset to exit these low-power modes.

When the CPU is inactive, a software reset cannot generate to exit any low-power mode.

9.3.4.6 System Control Module (SCM)

The SCM's core watchdog timer can bring the device out of all low-power modes except stop mode. In stop mode, all clocks stop, and the core watchdog timer does not operate.

When enabled, the core watchdog can bring the device out of low-power mode in one of two ways. Depending on the setting of the CWCR[CWRI] field, a core watchdog timeout may reset the device. Other settings of the CWRI field may enable a core watchdog interrupt and upon a watchdog timeout, this interrupt can bring the device out of low-power mode. This system setup must meet the conditions specified in Section 9.3.3, "Low-Power Modes," for the core watchdog interrupt to bring the part out of low-power mode.

9.3.4.7 GPIO Ports

The GPIO ports are unaffected by entry into a low-power mode. These pins may impact low-power current draw if they are configured as outputs and are sourcing current to an external load. If low-power mode is exited by a reset, the state of the I/O pins reverts to their default direction settings.

9.3.4.8 Interrupt Controllers (INTC0, INTC1)

The interrupt controller is not affected by any of the low-power modes. All logic between the input sources and generating the interrupt to the processor is combinational to allow the ability to wake up the core during low-power stop mode when all system clocks stop.

An interrupt request causes the processor to exit a low-power mode only if that interrupt’s priority level is at or above the level programmed in the interrupt priority mask field of the CPU’s status register (SR) and above the level programmed in the WCR[PRILVL]. The interrupt must also be enabled in the interrupt controller’s interrupt mask register as well as at the module from which the interrupt request would originate.

9.3.4.9 Edge Port

In wait and doze modes, the edge port continues to operate normally and may be configured to generate interrupts (either an edge transition or low level on an external pin) to exit the low-power modes.

In stop mode, no system clock is available to perform the edge detect function. Therefore, only the level detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

9.3.4.10 eDMA Controller

In wait and doze modes, the eDMA controller can bring the device out of a low-power mode by generating an interrupt upon completion of a transfer or upon an error condition. The completion of transfer interrupt generates when DMA interrupts are enabled by the setting of a EDMA_INTR[INT*n*], and an interrupt is generated when TCD*n*[DONE] is set. The interrupt upon error condition is generated when EDMA_EEIR[EEI*n*] is set, and an interrupt generates when any of the EDMA_ESR bits become set.

The eDMA controller is stopped in stop mode and thus cannot cause an exit from this low-power mode.

9.3.4.11 FlexBus Module

In wait and doze modes, the FlexBus module continues operation but does not generate interrupts; therefore, it cannot bring a device out of a low-power mode. This module is stopped in STOP mode.

9.3.4.12 SDRAM Controller (SDRAMC)

SDRAM controller operation is unaffected either the wait or doze modes; however, the SDRAMC is disabled by stop mode. Because the STOP mode disables all clocks to the SDRAMC, the SDRAMC does not generate refresh cycles.

To prevent data loss, SDRAMC should be placed in self-refresh mode by clearing SDCR[CKE] and setting SDCR[REF_EN]. The SDRAM self-refresh mode allows the SDRAM to enter a low-power state where internal refresh operations maintain the integrity of the SDRAM data.

When stop mode is exited, setting the SDCR[CKE] bit causes the SDRAM controller to exit the self-refresh mode and allow bus cycles to the SDRAM to resume.

NOTE

> The SDRAM is inaccessible in the self-refresh mode. Therefore, if stop mode is used, the vector table and any interrupt handlers that could wake the processor should not be stored in or attempt to access SDRAM.

9.3.4.13 Fast Ethernet Controller (FEC)

In wait and doze modes, the FEC is unaffected and may generate an interrupt to exit these low-power modes. In stop mode, the FEC stops immediately and freezes operation, register values, state machines, and external pins. The FEC clocks also shut down in this mode. Exiting stop mode returns the FEC to operation from the state prior to stop mode entry.

9.3.4.14 USB On-the-Go Module

If the USB On-the-Go module is clocked externally, it operates normally in wait and doze. It is capable of generating an interrupt to wake up the core from the wait and doze modes. In stop mode, the USB module is disabled.

The USB block contains an automatic low power mode in which the module enters suspend mode after a 6.0 ms minimum period of inactivity. When the module receives a wake-up from the USB host, the transceiver is re-enabled for normal USB operations.

9.3.4.15 PCI Controller

In wait and doze modes, the PCI controller is unaffected and may generate an interrupt to exit these low-power modes.

9.3.4.16 ATA Controller

In wait and doze modes, the ATA controller is unaffected and may generate an interrupt to exit these low-power modes.

9.3.4.17 Real Time Clock

In stop mode, the external clock driving EXTAL32K/XTAL32K continues to clock the RTC module. Therefore, the device can update the RTC counters, alarms, etc. while in stop mode. An RTC interrupt/wake-up can be generated while in stop mode to wakeup the device if the RTC alarms are triggered.

9.3.4.18 Programmable Interrupt Timers (PIT0–3)

In stop mode (or doze mode, if so programmed in the PCSR*n* register), the programmable interrupt timer (PIT) ceases operation, and freezes at the current value. When exiting these modes, the PIT resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the PIT may generate an interrupt to exit the low-power modes.

9.3.4.19 DMA Timers (DTIM0–3)

In wait and doze modes, the DMA timers may generate an interrupt to exit a low-power mode. This interrupt can generate when the DMA timer is in input capture mode or reference compare mode.

In input capture mode, where the capture enable (CE) field of the timer mode register (DTMR) has a non-zero value and the DTXMR[DMAEN] is cleared, an interrupt issues upon a captured input. In reference compare mode, where the output reference requests interrupt enable (ORRI) bit of DTMR is set and DTXMR[DMAEN] is cleared, an interrupt issues when the timer counter reaches the reference value.

DMA timer operation disables in stop mode. Upon exiting stop mode, the timer resumes operation unless stop mode was exited by reset.

9.3.4.20 DMA Serial Peripheral Interface (DSPI)

In wait and doze modes, the DSPI module is unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the DSPI stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the DSPI clocks shut down. Coming out of stop mode returns the DSPI to operation from the state prior to stop mode entry.

9.3.4.21 UART Modules (UART0–2)

In wait and doze modes, the UARTs are unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the UARTs stop immediately and freeze their operation, register values, state machines, and external pins. During this mode, the UART clocks shut down. Exiting stop mode returns the UARTs to the operation of the state prior to stop-mode entry.

9.3.4.22 I^{2}C Module

When the I^{2}C Module is enabled by the setting of the I2CR[IEN] bit and the device is not in stop mode, the I^{2}C module is operable and may generate an interrupt to bring the device out of a low-power mode. For an interrupt to occur, the I2CR[IIE] bit must be set to enable interrupts, and the setting of the I2SR[IIF] generates the interrupt signal to the CPU and interrupt controller. The setting of I2SR[IIF] signifies the completion of one byte transfer or the reception of a calling address matching its own specified address when in slave-receive mode.

In stop mode, the I^{2}C module stops immediately and freezes operation, register values, and external pins. Upon exiting stop mode, the I^{2}C resumes operation unless stop mode was exited by reset.

9.3.4.23 BDM

Entering halt (debug) mode via the BDM port (by asserting the external $\overline{\text{BKPT}}$ pin) causes the processor to exit any low-power mode.

9.3.4.24 JTAG

The JTAG (Joint Test Action Group) controller logic is clocked using the TCLK input and not affected by the system clock. The JTAG cannot generate an event to cause the processor to exit any low-power mode. Toggling TCLK during any low-power mode increases the system current consumption.

9.3.5 Summary of Peripheral State During Low-power Modes

The functionality of each of the peripherals and CPU during the various low-power modes is summarized in Table 9-9. The status of each peripheral during a given mode refers to the condition the peripheral automatically assumes when the STOP instruction is executed and the WCR[LPMD] field is set for the particular low-power mode. Individual peripherals may be disabled by programming its dedicated control bits. The wake-up capability field refers to the ability of an interrupt or reset by that peripheral to force the CPU into run mode.

Table 9-9. CPU and Peripherals in Low-Power Modes

Module	Peripheral Status[1] / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
ColdFire Core	Stopped	N/A	Stopped	N/A	Stopped	N/A
SRAM	Stopped	N/A	Stopped	N/A	Stopped	N/A
Clock Module	Enabled	Interrupt	Enabled	Interrupt	Program	Interrupt
Power Management	Enabled	N/A	Enabled	N/A	Stopped	N/A
Chip Configuration Module	Enabled	N/A	Enabled	N/A	Stopped	N/A
Reset Controller	Enabled	Reset	Enabled	Reset	Stopped	Reset
System Control Module	Enabled	Reset	Enabled	Reset	Stopped	N/A
GPIO	Enabled	N/A	Enabled	N/A	Stopped	N/A
Interrupt controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
Edge port	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
eDMA Controller	Enabled	Yes	Enabled	Yes	Stopped	N/A
FlexBus Module	Enabled	N/A	Enabled	N/A	Stopped	N/A
SDRAM Controller	Enabled	N/A	Enabled	N/A	Stopped	N/A
Fast Ethernet Controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
USB OTG	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
PCI Controller and Arbiter	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
ATA Controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A

Table 9-9. CPU and Peripherals in Low-Power Modes (continued)

Module	Peripheral Status[1] / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
SSI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
Real Time Clock	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
Programmable Interrupt Timers	Enabled	Interrupt	Program	Interrupt	Stopped	N/A
DMA Timers	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
DSPI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
UARTs	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
I^2C Module	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
RNG	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
JTAG[2]	Enabled	N/A	Enabled	N/A	Enabled	N/A
BDM[3]	Enabled	Yes	Enabled	Yes	Enabled	Yes

[1] Program indicates that the peripheral function during the low-power mode is dependent on programmable bits in the peripheral register map.

[2] The JTAG logic is clocked by a separate TCLK clock.

[3] Entering halt mode via the BDM port exits any lower-power mode. Upon exit from halt mode, the previous low-power mode is re-entered, and changes made in halt mode remain in effect.

Chapter 10
Universal Serial Bus Interface – On-The-Go Module

10.1 Introduction

This chapter describes the universal serial bus (USB) interface, which implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, you should refer to the governing specifications. Readers of this chapter are assumed to be fluent in the operation and requirements of a USB network.

Visit the USB Implementers Forum web page at http://www.usb.org/developers/docs for:

- *Universal Serial Bus Specification, Revision 2.0*
- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

Visit the Intel USB specifications web page at http://www.intel.com/technology/usb/spec.htm for:

- *Enhanced Host Controller Interface Specification for Universal Serial Bus, Revision 1.0*

Visit the ULPI web page at http://www.ulpi.org for:

- *UTMI+ Specification, Revision 1.0*
- *UTMI Low Pin Interface (ULPI) Specification, Revision 1.0*

10.1.1 Overview

The USB On-The-Go (OTG) module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The USB OTG module can act as a host, a device, or an On-The-Go negotiable host/device on the USB bus.

The USB 2.0 OTG module interfaces to the processor’s ColdFire core. The USB controller is programmable to support host, or device operations under firmware control. Full-speed (FS) and low-speed (LS) applications are supported by the integrated on-chip transceiver. The ULPI interface option supports high-speed (HS) applications. The processor’s on-chip PLL provides all necessary clocks to the USB controller, including a system interface clock and a 60 MHz clock. For special applications, pin access (via USBCLKIN) is provided for an external 60 MHz reference clock.

The USB controller provides control and status signals to interface with external USB OTG and USB host power devices. Use these control and status signals on the chip interface and the I^{2}C bus to communicate with external USB On-The-Go and USB host power devices.

USB-host modules must supply 500 mA with a 5 V supply on its downstream port (referred to as VBUS); however, the USB OTG standard provides a minimum 8 mA VBUS supply requirement. Optionally, the OTG module may supply up to 500 mA to the USB-connected devices. If the connected device attempts to draw more than the allocated amount of current, the USB host must disable the port and remove power. USB VBUS is not provided on-chip. This processor provides pins for control and status to an external IC capable of managing the VBUS downstream supply.

For OTG operations, external circuitry is required to manage the host negotiation protocol (HNP) and session request protocol (SRP). External ICs that are capable of providing the OTG VBUS with support for HNP and SRP, as well as support for programmable pullup and pulldown resistors on the USB DP and DM lines are available from various manufacturers.

The on-chip FS/LS transceiver also includes a programmable pullup resistor on USB DP. This pullup is configurable via the CCM. See Chapter 11, “Chip Configuration Module (CCM),” for more information. The primary function of the transceiver is the physical signal conditioning of the external USB DP and DM cable signals for a USB 2.0 network. Several USB system elements are not supported on the device as they are available via standard products from various manufacturers.

10.1.2 Block Diagram

Figure 10-1 shows the USB On-The-Go interface using the on-chip full-speed/low-speed transceiver.

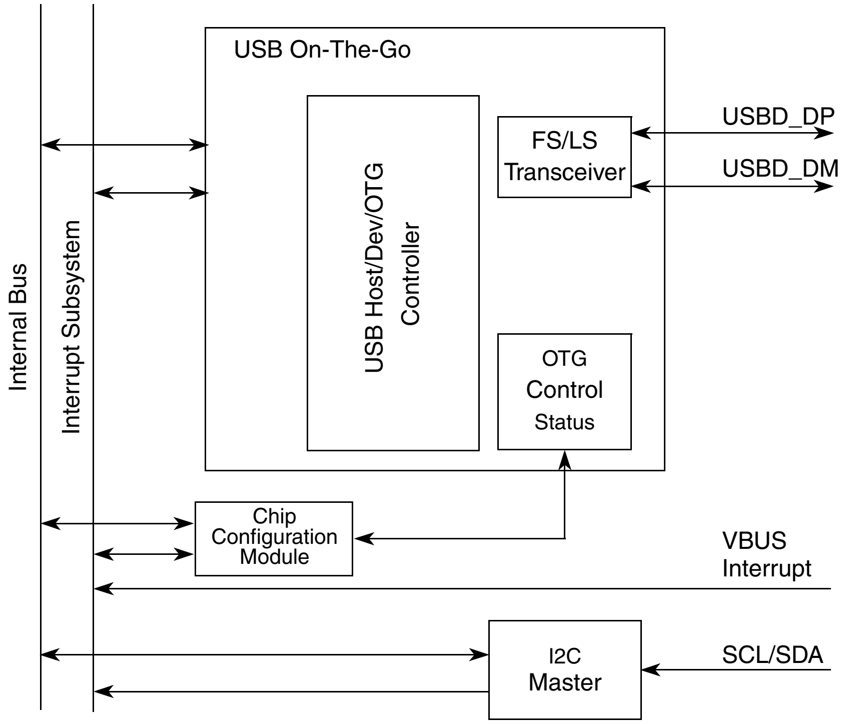


Figure 10-1. USB On-The-Go with on-chip FS/LS Transceiver Interface Block Diagram

Figure 10-2 illustrates the On-The-Go (OTG) configuration with an off-chip ULPI transceiver. The ULPI transceiver is an implementation of the HS/FS/LS physical layer which encapsulates the 60+ pin UTMI+ interface using a 12-pin digital interface.

The board-level implementation of a ULPI based product is dependent on the PHY vendor. One possible implementation is shown in Figure 10-1. The ULPI PHY manages USB clocking, DP/DM bias resistors, and the OTG VBUS charge pump. For OTG applications requiring full host power (100 – 500 mA downstream current), an additional USB power-switch chip may be used. This OTG configuration may be used as a USB device, host, or dual-role device under firmware control.

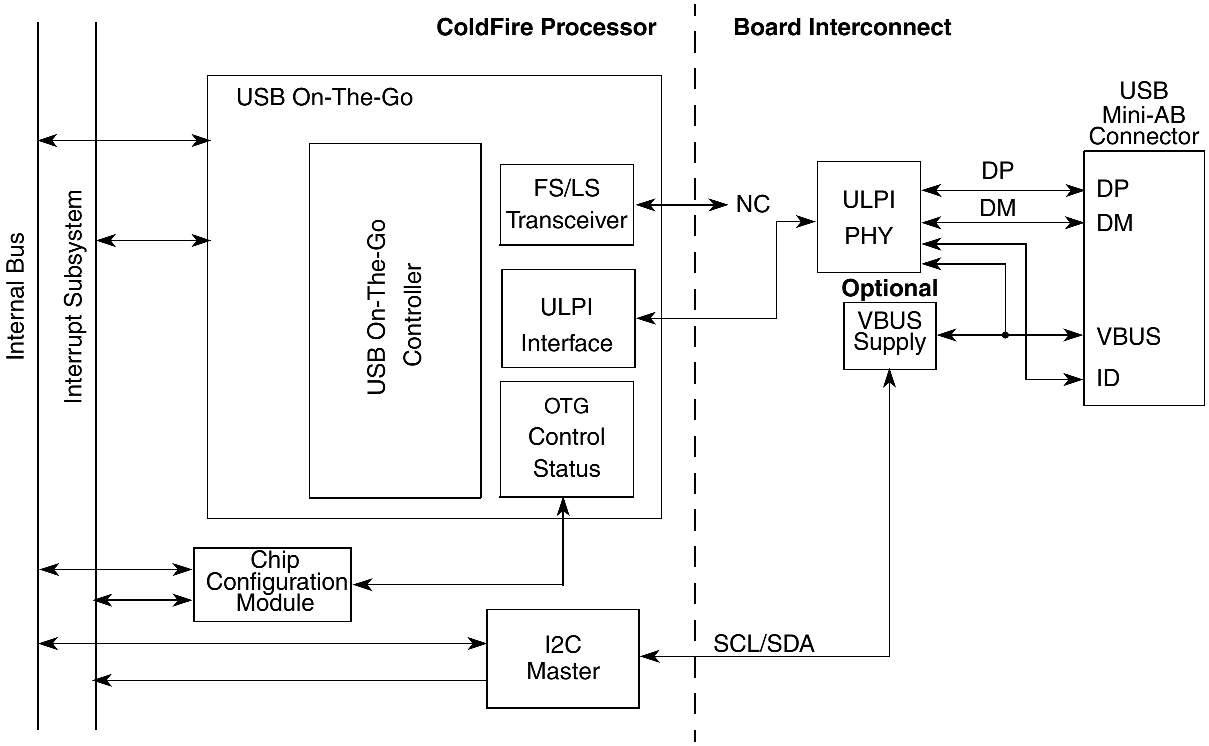


Figure 10-2. USB On-The-Go module and ULPI transceiver/PHY

10.1.3 Features

The USB On-The-Go module includes these features:

- Complies with USB specification rev 2.0
- USB host mode
  - — Supports enhanced-host-controller interface (EHCI).
  - — Allows direct connection of FS/LS devices without an OHCI/UHCI companion controller.
  - — Supported by Linux and other commercially available operating systems.
- USB device mode
  - — Supports full-speed operation via the on-chip transceiver.
  - — Supports full-speed/high-speed operation via an external ULPI transceiver.
  - — Supports one upstream facing port.
  - — Supports four programmable, bidirectional USB endpoints, including endpoint 0. See endpoint configurations:

Table 10-1. Endpoint Configurations

Endpoint	Type	FIFO Size	Data Transfer	Comments
0	Bidirectional	Variable	Control	Mandatory
1-3	IN or OUT	Variable	Ctrl, Int, Bulk, or Iso	Optional

- Suspend mode/low power
  - As host, firmware can suspend individual devices or the entire USB and disable chip clocks for low-power operation
  - Device supports low-power suspend
  - Remote wake-up supported for host and device
  - Integrated with the processor's doze and stop modes for low power operation
- Includes an on-chip full-speed (12 Mbps) and low-speed (1.5 Mbps) transceiver
- Support for off-chip HS/FS/LS transceiver
  - External ULPI transceiver supports high speed (480 Mbps), full speed, and low speed operation in host mode, and high-speed and full-speed operation in device mode
  - Interface uses 8-bit single-data-rate ULPI data bus
  - ULPI PHY supplies a 60 MHz USB reference clock input to the processor

10.1.4 Modes of Operation

The USB OTG module has two basic operating modes: host and device. Selection of operating mode is accomplished via the USBMODE[CM] bit field.

Speed selection is auto-detected at connect time via sensing of the DP or DM pull-up resistor on the connected device using enumeration procedures in the USB network. The USB OTG module provides these operation modes:

- USB disabled. In this mode, the USB OTG's datapath does not accept transactions received on the USB interface.
- USB enabled. In this mode, the USB host's datapath is enabled to accept transactions received on the USB interface.
- USB enabled, low-power modes. See Section 10.1.4.1, "Low-Power Modes," for details.

10.1.4.1 Low-Power Modes

The USB OTG module is integrated with the processor's low-power modes (stop, doze and wait). The modes are implemented as follows:

- Stop — The processor stops the clock to the USB OTG module. In this state, the USB OTG module ignores traffic on the USB and does not generate any interrupts or wake-up events. The on-chip transceiver is disabled to save power.
- Wait — The clocks to the USB OTG module are running.

- Doze — The processor stops the system clocks to the USB OTG module, but the 60 MHz transceiver clock remains active. In doze mode, detection of resume signaling initiates a restart of the module clocks.

10.2 External Signal Description

Table 10-2 describes the external signal functionality of the USB OTG module.

NOTE

The ULPI signals are multiplexed with the FEC module. This section describes the signal functions when in ULPI mode; refer to Chapter 16, "Pin Multiplexing and Control," for more details.

Table 10-2. USB OTG Signal Descriptions

Signal	I/O	Description	
On-chip FS/LS transceiver			
USB_CLKIN	I	Optional 60 MHz clock source. This signal is also used for the input clock from a ULPI PHY.	
USB_DM	I/O	Data minus. Output of dual-speed transceiver for the USB OTG module.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous
USB_DP	I/O	Data plus. Output of dual-speed transceiver for the USB OTG module.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous
USB_PULLUP	O	Enables an external pull-up on the USB_DP line. This signal is controlled by the UOCSR[BVLD] bit.	
		State Meaning	Asserted—Pull-up enabled. UOCSR[BVLD] set. Negated—Pull-up disabled. UOCSR[BVLD] cleared.
		Timing	Asynchronous
ULPI Interface			
ULPI_DIR	I	Direction. ULPI_DIR controls data bus direction. When PHY has data to transfer to USB port, it drives ULPI_DIR high to take ownership of the bus. When the PHY has no data to transfer, it drives ULPI_DIR low and monitors the bus for link activity. The PHY pulls ULPI_DIR high when the interface cannot accept data from the link. For example, when PHY's PLL is not stable.	
		State Meaning	Asserted—PHY has data to transfer to the link. Negated—PHY has no data to transfer.
		Timing	Synchronous to USB_CLKIN or ULPI_CLK.

Table 10-2. USB OTG Signal Descriptions (continued)

Signal	I/O	Description	
ULPI_NXT	I	Next data. PHY asserts ULPI_NXT to throttle data. When USB port sends data to the PHY, ULPI_NXT indicates when PHY accepts the current byte. The USB port places the next byte on the data bus in the following clock cycle. When the PHY sends data to USB port, ULPI_NXT indicates when a new byte is available for USB port to consume.	
		State Meaning	Asserted—PHY is ready to transfer byte. Negated—PHY is not ready.
		Timing	Synchronous to ULPI_CLK.
ULPI_STP	O	Stop. ULPI_STP indicates the end of a transfer on the bus.	
		State Meaning	Asserted—USB asserts this signal for one clock cycle to stop the data stream currently on the bus. If the USB port sends data to the PHY, ULPI_STP indicates the last data byte was previously on the bus. If the PHY is sending data to the USB port, ULPI_STP forces the PHY to end its transfer, deassert ULPI_DIR, and relinquish control of the data bus to the USB port. Negated—Indicates normal operation.
		Timing	Synchronous to USB_CLK or ULPI_CLK
ULPI_DATA[7:0]	I/O	Data bit n. ULPI_DATAT*n* is bit *n* of the 8-bit, bi-directional data bus used to carry USB, register, and interrupt data between the USB port controller and the PHY.	
		State Meaning	Asserted—Data bit *n* is 1. Negated—Data bit *n* is 0.
		Timing	Synchronous to USB_CLK or ULPI_CLK

10.2.1 USB OTG Control and Status Signals

The USB OTG module uses a number of control and status signals to implement the OTG protocols. The USB OTG module must be able to individually enable and disable the pull-up and pull-down resistors on DP and DM, and it must be able to control and sense the levels on the USB VBUS line.

These control and status signals are implemented on chip as registers within the chip-configuration module (CCM) to minimize the pin-count on the device. With firmware, the system designer uses an external device to manage the OTG functions to implement communications across the I^{2}C bus or GPIO pins.

The OTG controller status register (UOCSR) implements as follows:

- Writes to the UOCSR register from the firmware set the corresponding bits on the USB interface.
- When the USB OTG module outputs change, the corresponding bits on the UOCSR register are updated, and a maskable interrupt is generated.

The UOCSR register is documented in the CCM chapter, see Section 11.3.6, “USB On-the-Go Controller Status Register (UOCSR).”

Table 10-3. Internal Control and Status Bits for USB OTG Module

Signal	Mnemonic	Direction	Comments	Interrupt Trigger?
DP Pull-down Enable	DPPD	Enables 15 kΩ resistor pull-down on DP	R	Y
DM Pull-down Enable	DMPD	Enables 15 kΩ resistor pull-down on DM	R	Y
VBUS Charge	CRG_VBUS	Enables 8 mA pull-up to charge VBUS.	R	Y
VBUS Discharge	DCR_VBUS	Enables 8 mA pull-down to discharge VBUS.	R	Y
DP Pull-up Enable	DPPU	Enables the 1.5KΩ resistor pull-up on DP	R	Y
A Session Valid	AVLD	Indicates a valid session level for A device detected on VBUS.	R/W	N
B Session Valid	BVLD	Indicates a valid session level for B device detected on VBUS.	R/W	N
Session Valid	VVLD	Indicates valid operating level on VBUS from USB device's perspective.	R/W	N
Session End	SEND	Indicates VBUS fell below the session valid threshold.	R/W	N
VBUS Fault	PWRFLT	Indicates a fault (overcurrent, thermal issue) on VBUS.	R/W	N
Wake-up Event	WKUP	Reflects when a wake-up event occurred on the USB bus.	R/W	Y
Interrupt Mask	UOMIE	Interrupt enable. When this bit is 1, changes on DPPD, DMPD, DPPU, CHRG_VBUS, DCRG_VBUS, or VBUS_PWR cause an interrupt to be asserted. When this bit is 0, the interrupt is masked.	R/W	N/A
On-chip Transceiver Pull-down Enable	XPDE	Enables the on-chip 50 kΩ pull-downs on the OTG controller's DM and DP pins when the on-chip transceiver is used.	R/W	N

10.3 Memory Map/Register Definition

This section provides the memory map and detailed descriptions of all USB-interface registers. See Table 10-4 for the memory map of the USB OTG interface.

Table 10-4. USB On-The-Go Memory Map

Address	Register	EHCI[1]	H/D[2]	Width (bits)	Access	Reset	Section/Page
Module Identification Registers							
0xFC0B_0000	Identification Register (ID)	N	H/D	32	R	0x0042_FA05	10.3.1.1/10-9
0xFC0B_0004	General Hardware Parameters (HWGENERAL)	N	H/D	32	R	0x0000_07C5	10.3.1.2/10-10
0xFC0B_0008	Host Hardware Parameters (HWHOST)	N	H/D	32	R	0x1002_0001	10.3.1.3/10-11
0xFC0B_000C	Device Hardware Parameters (HWDEVICE)	N	D	32	R	0x0000_0009	10.3.1.4/10-11
0xFC0B_0010	TX Buffer Hardware Parameters (HWTXBUF)	N	H/D	32	R	0x8004_0604	10.3.1.5/10-12
0xFC0B_0014	RX Buffer Hardware Parameters (HWRXBUF)	N	H/D	32	R	0x0000_0404	10.3.1.6/10-12
Device/Host Timer Registers							
0xFC0B_0080	General Purpose Timer 0 Load (GPTIMER0LD)	N	H/D	32	R/W	0x0000_0000	10.3.2.1/10-13
0xFC0B_0084	General Purpose Timer 0 Control (GPTIMER0CTL)	N	H/D	32	R/W	0x0000_0000	10.3.2.2/10-13
0xFC0B_0088	General Purpose Timer 1 Load (GPTIMER1LD)	N	H/D	32	R/W	0x0000_0000	10.3.2.1/10-13
0xFC0B_008C	General Purpose Timer 1 Control (GPTIMER1CTL)	N	H/D	32	R/W	0x0000_0000	10.3.2.2/10-13
Capability Registers							
0xFC0B_0100	Host Interface Version Number (HCIVERSION)	Y	H	16	R	0x0100	10.3.3.1/10-14
0xFC0B_0103	Capability Register Length (CAPLENGTH)	Y	H/D	8	R	0x40	10.3.3.2/10-15
0xFC0B_0104	Host Structural Parameters (HCSPARAMS)	Y	H	32	R	0x0001_0011	10.3.3.3/10-15
0xFC0B_0108	Host Capability Parameters (HCCPARAMS)	Y	H	32	R	0x0000_0006	10.3.3.4/10-16
0xFC0B_0122	Device Interface Version Number (DCIVERSION)	N	D	16	R	0x0001	10.3.3.5/10-17
0xFC0B_0124	Device Capability Parameters (DCCPARAMS)	N	D	32	R	0x0000_0184	10.3.3.6/10-17
Operational Registers							
0xFC0B_0140	USB Command (USBCMD)	Y	H/D	32	R/W	0x0008_0000	10.3.4.1/10-18
0xFC0B_0144	USB Status (USBSTS)	Y	H/D	32	R/W	0x0000_0080	10.3.4.2/10-20
0xFC0B_0148	USB Interrupt Enable (USBINTR)	Y	H/D	32	R/W	0x0000_0000	10.3.4.3/10-23
0xFC0B_014C	USB Frame Index (FRINDEX)	Y	H/D	32	R/W	0x0000_0000	10.3.4.4/10-25
0xFC0B_0154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	H	32	R/W	0x0000_0000	10.3.4.5/10-26
0xFC0B_0154	Device Address (DEVICEADDR)	N	D	32	R/W	0x0000_0000	10.3.4.6/10-27
0xFC0B_0158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	H	32	R/W	0x0000_0000	10.3.4.7/10-27
0xFC0B_0158	Address at Endpoint List (EPLISTADDR)	N	D	32	R/W	0x0000_0000	10.3.4.8/10-28
0xFC0B_015C	Host TT Asynchronous Buffer Control (TTCTRL)	N	H	32	R/W	0x0000_0000	10.3.4.9/10-28
0xFC0B_0160	Master Interface Data Burst Size (BURSTSIZE)	N	H/D	32	R/W	0x0000_0404	10.3.4.10/10-29
0xFC0B_0164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	H	32	R/W	0x0000_0000	10.3.4.11/10-29

Table 10-4. USB On-The-Go Memory Map (continued)

Address	Register	EHCI[1]	H/D[2]	Width (bits)	Access	Reset	Section/Page
0xFC0B_0170	ULPI Register Access (ULPI_VIEWPORT)	N	H/D	32	R/W	0x0000_0000	10.3.4.12/10-31
0xFC0B_0180	Configure Flag Register (CONFIGFLAG)	Y	H/D	32	R	0x0000_0001	10.3.4.13/10-33
0xFC0B_0184	Port Status/Control (PORTSC1)	Y	H/D	32	R/W	0xEC00_0004	10.3.4.14/10-33
0xFC0B_01A4	On-The-Go Status and Control (OTGSC)	N	H/D	32	R/W	0x0000_1020	10.3.4.15/10-37
0xFC0B_01A8	USB Mode Register (MODE)	N	H/D	32	R/W	0x0000_0000	10.3.4.16/10-40
0xFC0B_01AC	Endpoint Setup Status Register (EPSETUPSR)	N	D	32	R/W	0x0000_0000	10.3.4.17/10-41
0xFC0B_01B0	Endpoint Initialization (EPPRIME)	N	D	32	R/W	0x0000_0000	10.3.4.18/10-41
0xFC0B_01B4	Endpoint De-initialize (EPFLUSH)	N	D	32	R/W	0x0000_0000	10.3.4.19/10-42
0xFC0B_01B8	Endpoint Status Register (EPSR)	N	D	32	R	0x0000_0000	10.3.4.20/10-42
0xFC0B_01BC	Endpoint Complete (EPCOMPLETE)	N	D	32	R/W	0x0000_0000	10.3.4.21/10-43
0xFC0B_01C0	Endpoint Control Register 0 (EPCR0)	N	D	32	R/W	0x0080_0080	10.3.4.22/10-44
0xFC0B_01C4	Endpoint Control Register 1 (EPCR1)	N	D	32	R/W	0x0000_0000	10.3.4.23/10-45
0xFC0B_01C8	Endpoint Control Register 2 (EPCR2)	N	D	32	R/W	0x0000_0000	10.3.4.23/10-45
0xFC0B_01CC	Endpoint Control Register 3 (EPCR3)	N	D	32	R/W	0x0000_0000	10.3.4.23/10-45

[1] Indicates if the register is present in the EHCI specification.

[2] Indicates if the register is available in host and/or device modes.

10.3.1 Module Identification Registers

Declare the slave interface presence and include a table of the hardware configuration parameters. These registers are not defined by the EHCI specification.

10.3.1.1 Identification (ID) Register

Provides a simple way to determine if the module is provided in the system. The ID register identifies the module and its revision.

Address: 0xFC0B_0000 (ID)

Access: User 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
R	0	0	0	0	0	0	0	0	REVISION								1	1	NID						0	0	ID					
W																																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1

Figure 10-3. Identification Register (ID)

Table 10-5. ID Field Descriptions

Field	Description
31–24	Reserved, always cleared.
23–16 REVISION	Revision number of the module.
15–14	Reserved, always set.
13–8 NID	Ones-complement version of the ID bit field.
7–6	Reserved, always cleared.
5–0 ID	Configuration number. This number is set to 0x05.

10.3.1.2 General Hardware Parameters Register (HWGENERAL)

The HWGENERAL register contains parameters defining the particular implementation of the module.

Address: 0xFC0B_0004 (HWGENERAL) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SM		PHYM			PHYW		0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	1	0	1

Figure 10-4. General Hardware Parameters Register (HWGENERAL)

Table 10-6. HWGENERAL Field Descriptions

Field	Description
31–11	Reserved, always cleared.
10–9 SM	Serial mode. Indicates presence of serial interface. Always 11. 11 Serial engine is present and defaulted for all FS/LS operations
8–6 PHYM	PHY Mode. Indicates USB transceiver interface used. Always reads 111. 111 Software controlled reset to serial FS
5–4 PHYW	PHY width. Indicates data interface to UTMI transceiver. This field is relevant only for UTMI mode; therefore, it is relevant only to the USB OTG module in UTMI mode. Always reads 00. 00 8-bit data bus (60 MHz)
3	Reserved, always cleared.
2–1	Reserved. For the USB OTG module, always 10.
0	Reserved, always set.

10.3.1.3 Host Hardware Parameters Register (HWHOST)

Provides host hardware parameters for this implementation of the module.

Address: 0xFC0B_0008 (HWHOST)

Access: User 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
R	TTPER								TTASY								0	0	0	0	0	0	0	0	0	0	0	0	NPORT			HC
W																																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 10-5. Host Hardware Parameters Register (HWHOST)

Table 10-7. HWHOST Field Descriptions

Field	Description
31–24 TTPER	Transaction translator periodic contexts. Number of supported transaction translator periodic contexts. Always 0x10. 0x10 16
23–16 TTASY	Transaction translator contexts. Number of transaction translator contexts. Always 0x02. 0x02 2
15–4	Reserved, always cleared.
3–1 NPORT	Indicates number of ports in host mode minus 1. Always 0 for the USB OTG module.
0 HC	Indicates module is host capable. Always set.

10.3.1.4 Device Hardware Parameters Register (HWDEVICE)

Provides device hardware parameters for this implementation of the USB OTG module.

Address: 0xFC0B_000C (HWDEVICE)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DEVEP					DC
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

Figure 10-6. Device Hardware Parameters Register (HWDEVICE)

Table 10-8. HWDEVICE Field Descriptions

Field	Description
31–6	Reserved, always cleared.
5–1 DEVEP	Device endpoints. The number of supported endpoints. Always 0x04.
0 DC	Indicates the OTG module is device capable. Always set.

10.3.1.5 Transmit Buffer Hardware Parameters Register (HWTXBUF)

Provides the transmit-buffer parameters for this implementation of the module.

Address: 0xFC0B_0010 (HWTXBUF) Access: User 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
R	TXLC	0	0	0	0	0	0	0	TXCHANADD								TXADD								TXBURST							
W																																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0

Figure 10-7. Transmit Buffer Hardware Parameters Register (HWTXBUF)

Table 10-9. HWTXBUF Field Descriptions

Field	Description
31 TXLC	Transmit local context registers. Indicates how the device transmit context registers implement. Always set on USB OTG module. 0 Store device transmit contexts in the TX FIFO 1 Store device transmit contexts in a register file
30–24	Reserved, always cleared.
23–16 TXCHANADD	Transmit channel address. Number of address bits required to address one channel's worth of TX data. Always 0x04.
15–8 TXADD	Transmit address. Number of address bits for the entire TX buffer. Always 0x06.
7–0 TXBURST	Transmit burst. Indicates number of data beats in a burst for transmit DMA data transfers. Always 0x04.

10.3.1.6 Receive Buffer Hardware Parameters Register (HWRXBUF)

Provides the receive buffer parameters for this implementation of the module.

Address: 0xFC0B_0014 (HWRXBUF) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RXADD								RXBURST							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 10-8. Receive Buffer Hardware Parameters Register (HWRXBUF)

Table 10-10. HWRXBUF Field Descriptions

Field	Description
31–16	Reserved.
15–8 RXADD	Receive address. The number of address bits for the entire RX buffer. Always 0x04.
7–0 RXBURST	Receive burst. Indicates the number of data beats in a burst for receive DMA data transfers. Always 0x04.

10.3.2 Device/Host Timer Registers

The host/device controller drivers can measure time-related activities using these timer registers, which are not defined by the EHCI specification.

10.3.2.1 General Purpose Timer *n* Load Registers (GPTIMER*n*LD)

The GPTIMER*n*LD registers contain the timer duration or load value.

Address: 0xFC0B_0080 (GPTIMER0LD)
0xFC0B_0088 (GPTIMER1LD)

Access: User 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
R	0	0	0	0	0	0	0	0	GPTLD																							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-9. General Purpose Timer *n* Load Registers (GPTIMER*n*LD)

Table 10-11. GPTIMER*n*LD Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–0 GPTLD	Specifies the value to be loaded into the countdown timer on a reset. The value in this register represents the time in microseconds minus 1 for the timer duration. For example, for a one millisecond timer, load 1000 – 1 = 999 (0x00_03E7). **Note:** Maximum value of 0xFF_FFFF or 16.777215 seconds.

10.3.2.2 General Purpose Timer *n* Control Registers (GPTIMER*n*CTL)

The GPTIMER*n*CTL registers control the various functions of the general purpose timers.

Address: 0xFC0B_0084 (GPTIMER0CTL)
0xFC0B_008C (GPTIMER1CTL)

Access: User 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
R	RUN	0	0	0	0	0	0	MODE	GPTCNT																							
W		RST																														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-10. General Purpose Timer *n* Control Registers (GPTIMER*n*CTL)

Table 10-12. GPTIMER*n*CTL Field Descriptions

Field	Description
31 RUN	Timer run. Enables the general purpose timer. Setting or clearing this bit does not have an effect on the GPTCNT field. 0 Timer stop 1 Timer run
30 RST	Timer reset. Setting this bit reloads GPTCNT with the value in GPTIMER*n*LD[GPTLD]. 0 No action 1 Load counter value

Table 10-12. GPTIMER*n*CTL Field Descriptions (continued)

Field	Description
29–25	Reserved, must be cleared.
24 MODE	Timer mode. Selects between a single timer countdown and a looped countdown. In one-shot mode, the timer counts down to zero, generates an interrupt, and stops until the counter is reset by software. In repeat mode, the timer counts down to zero, generates an interrupt, and automatically reloads the counter and begins another countdown. 0 One shot 1 Repeat
23–0 GPTCNT	Timer count. Indicates the current value of the running timer.

10.3.3 Capability Registers

Specifies software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers not defined by the EHCI specification are noted in their descriptions.

10.3.3.1 Host Controller Interface Version Register (HCIVERSION)

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this OTG controller. The most-significant byte of the register represents a major revision; the least-significant byte is the minor revision. Figure 10-11 shows the HCIVERSION register.

Address: 0xFC0B_0100 (HCIVERSION)

Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HCIVERSION															
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 10-11. Host Controller Interface Version Register (HCIVERSION)

Table 10-13. HCIVERSION Field Descriptions

Field	Description
15–0 HCIVERSION	EHCI revision number. Value is 0x0100 indicating version 1.0.

10.3.3.2 Capability Registers Length Register (CAPLENGTH)

Register is used as an offset to add to the register base address to find the beginning of the operational register space, the location of the USBCMD register.

Address: 0xFC0B_0103 (CAPLENGTH) Access: User read-only

	7	6	5	4	3	2	1	0
R	CAPLENGTH							
W								
Reset:	0	1	0	0	0	0	0	0

Figure 10-12. Capability Registers Length Register (CAPLENGTH)

Table 10-14. CAPLENGTH Field Descriptions

Field	Description
7–0 CAPLENGTH	Capability registers length. Always 0x40.

10.3.3.3 Host Controller Structural Parameters Register (HCSPARAMS)

This register contains structural parameters such as the number of downstream ports. Figure 10-13 shows the HCSPARAMS register.

Address: 0xFC0B_0104 (HCSPARAMS) Access: User 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
R	0	0	0	0	N_TT				N_PTT				0	0	0	PI	N_CC				N_PCC				0	0	0	PPC	N_PORTS			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Figure 10-13. Host Controller Structural Parameters Register (HCSPARAMS)

Table 10-15. HCSPARAMS Field Descriptions

Field	Description
31–28	Reserved, always cleared.
27–24 N_TT	Number of transaction translators. Non-EHCI field. Indicates number of embedded transaction translators associated with host controller. This field is always 0x0. See Section 10.5.5.1, “Embedded Transaction Translator Function,” for more information on embedded transaction translators.
23–20 N_PTT	Ports per transaction translator. Non-EHCI field. Indicates number of ports assigned to each transaction translator within host controller.
19–17	Reserved, always cleared.
16 PI	Port indicators. Indicates whether the ports support port indicator control. Always cleared. 0 No port indicator fields. 1 The port status and control registers include a R/W field for controlling the state of the port indicator. See Table 10-3 for more information.

Table 10-15. HCSPARAMS Field Descriptions (continued)

Field	Description
15–12 N_CC	Number of companion controllers. Indicates number of companion controllers associated with USB OTG controller. Always cleared.
11–8 N_PCC	Number ports per CC. Indicates number of ports supported per internal companion controller. This field is 0 because no companion controllers are present.
7–5	Reserved, always cleared.
4 PPC	Power port control. Indicates whether host controller supports port power control. Always set. 1 Ports have power port switches.
3–0 N_PORTS	Number of ports. Indicates number of physical downstream ports implemented for host applications. Field value determines how many addressable port registers in the operational register. For the USB OTG module, this is always 0x1.

10.3.3.4 Host Controller Capability Parameters Register (HCCPARAMS)

Identifies multiple mode control (time-base bit functionality) addressing capability.

Address: 0xFC0B_0108 (HCCPARAMS) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EECP								IST				0	ASP	PFL	ADC
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 10-14. Host Controller Capability Parameters Register (HCCPARAMS)

Table 10-16. HCCPARAMS Field Descriptions

Field	Description
31–16	Reserved, always cleared.
15–8 EECP	EHCI extended capabilities pointer. This optional field indicates the existence of a capabilities list. 0x00 No extended capabilities are implemented. This field is always 0.
7–4 IST	Isochronous scheduling threshold. Indicates where software can reliably update the isochronous schedule, relative to the current position of the executing host controller. This field is always 0. 0 The value of the least significant 3 bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state.
3	Reserved, always cleared.
2 ASP	Asynchronous schedule park capability. Indicates if the host controller supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. This bit is always set. 0 Park not supported. 1 Park supported.

Table 10-16. HCCPARAMS Field Descriptions (continued)

Field	Description
1 PFL	Programmable frame list flag. Indicates that system software can specify and use a frame list length less that 1024 elements. This bit is always set. 1 Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4K-page boundary. This requirement ensures that the frame list is always physically contiguous.
0 ADC	64-bit addressing capability. This field is always 0; 64-bit addressing is not supported. 0 Data structures use 32-bit address memory pointers

10.3.3.5 Device Controller Interface Version (DCIVERSION)

Not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision.

Address: 0xFC0B_0122 (DCIVERSION) Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DCIVERSION															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 10-15. Device Controller Interface Version Register (DCIVERSION)

Table 10-17. DCIVERSION Field Descriptions

Field	Description
15–0 DCIVERSION	Device interface revision number.

10.3.3.6 Device Controller Capability Parameters (DCCPARAMS)

Not defined in the EHCI specification. Register describes the overall host/device capability of the USB OTG module.

Address: 0xFC0B_0124 (DCCPARAMS) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HC	DC	0	0	DEN				
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0

Figure 10-16. Device Control Capability Parameters (DCCPARAMS)

Table 10-18. DCCPARAMS Field Descriptions

Field	Description
31–9	Reserved, always cleared.
8 HC	Host capable. Indicates the USB OTG controller can operate as an EHCI compatible USB 2.0 host. Always set.

Table 10-18. DCCPARAMS Field Descriptions (continued)

Field	Description
7 DC	Device Capable. Indicates the USB OTG controller can operate as an USB 2.0 device. Always set.
6–5	Reserved, always cleared.
4–0 DEN	Device endpoint number. This field indicates the number of endpoints built into the device controller. Always 0x04.

10.3.4 Operational Registers

Comprised of dynamic control or status registers and are defined below.

10.3.4.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Address: 0xFC0B_0140 (USBCMD) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	ITC							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FS2	ATDTW	SUTW	0	ASPE	0	ASP		0	IAA	ASE	PSE	FS1	FS0	RST	RS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-17. USB Command Register (USBCMD)

Table 10-19. USBCMD Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–16 ITC	Interrupt threshold control. System software uses this field to set the maximum rate at which the module issueS interrupts. ITC contains maximum interrupt interval measured in microframes. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 64 microframes Else Reserved
15 FS2	See the FS bit description below. This is a non-EHCI bit.

Table 10-19. USBCMD Field Descriptions (continued)

Field	Description
14 ATDTW	Add dTD TripWire. This is a non-EHCI bit. This bit is used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit is also cleared by hardware when the state machine is in a hazard region where adding a dTD to a primed endpoint may go unrecognized. More information appears in Section 10.5.3.6.3, "Executing a Transfer Descriptor."
13 SUTW	Setup TripWire. A non-EHCI bit. Used as a semaphore to ensure that the setup data payload of 8 bytes is extracted from a QH by driver software without being corrupted. If the setup lockout mode is off (USBMODE[SLOM] = 1) then a hazard exists when new setup data arrives, and the software copies setup from the QH for a previous setup packet. This bit is set and cleared by software and is cleared by hardware when a hazard exists. More information appears in Section 10.5.3.4.4, "Control Endpoint Operation."
12	Reserved, must be cleared.
11 ASPE	Asynchronous schedule park mode enable. Software uses this bit to enable or disable park mode. 1 Park mode enabled 0 Park mode disabled
10	Reserved, must be cleared.
9–8 ASP	Asynchronous schedule park mode count. Contains a count of the successive transactions the host controller can execute from a high-speed queue head on the asynchronous schedule before continuing traversal of the asynchronous schedule. Valid values are 0x1 to 0x3. Software must not write a zero to this field when ASPE is set as this results in undefined behavior.
7	Reserved, must be cleared.
6 IAA	Interrupt on async advance doorbell. Used as a doorbell by software to tell controller to issue an interrupt the next time it advances the asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When controller has evicted all appropriate cached schedule states, it sets USBSTS[AAI] register. If the USBINTR[AAE] bit is set, the host controller asserts an interrupt at the next interrupt threshold. The controller clears this bit after it has set the USBSTS[AAI] bit. Software must not write a 1 to this bit when the asynchronous schedule is inactive. Doing so yields undefined results. This bit used only in host mode. Writing a 1 to this bit when the USB OTG module is in device mode has undefined results.
5 ASE	Asynchronous schedule enable. Controls whether the controller skips processing the asynchronous schedule. Only used in host mode. 1 Use the ASYNCLISTADDR register to access asynchronous schedule. 0 Do not process asynchronous schedule.
4 PSE	Periodic schedule enable. Controls whether the controller skips processing periodic schedule. Used only in host mode. 1 Use the PERIODICLISTBASE register to access the periodic schedule. 0 Do not process periodic schedule.
3–2 FS	Frame list size. With bit 15, these bits make the FS[2:0] fields, which specifies the frame list size controlling which bits in the frame index register must be used for the frame list current index. Used only in host mode. **Note:** Values below 256 elements are not defined in the EHCI specification. 000 1024 elements (4096 bytes) 001 512 elements (2048 bytes) 010 256 elements (1024 bytes) 011 128 elements (512 bytes) 100 64 elements (256 bytes) 101 32 elements (128 bytes) 110 16 elements (64 bytes) 111 8 elements (32 bytes)

Table 10-19. USBCMD Field Descriptions (continued)

Field	Description
1 RST	Controller reset. Software uses this bit to reset controller. Controller clears this bit when reset process completes. Clearing this register does not allow software to terminate the reset process early. Host mode: When software sets this bit, the controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction in progress on the USB immediately terminates. A USB reset is not driven on downstream ports. Software must not set this bit when the USBSTS[HCH] bit is cleared. Attempting to reset an actively running host controller results in undefined behavior. Device mode: When software sets this bit, the controller resets its internal pipelines, timers, counters, state machines, etc. to their initial value. Setting this bit with the device in the attached state is not recommended because it has an undefined effect on an attached host. To ensure the device is not in an attached state before initiating a device controller reset, all primed endpoints must be flushed and the USBCMD[RS] bit must be cleared.
0 RS	Run/Stop. Host mode: When set, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set. When this bit is cleared, the controller completes the current transaction on the USB and then halts. The USBSTS[HCH] bit indicates when the host controller finishes the transaction and enters the stopped state. Software must not set this bit unless controller is in halted state (USBSTS[HCH] = 1). Device mode: Setting this bit causes the controller to enable a pull-up on DP and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up becomes disabled upon transitioning into high-speed mode. Software must use this bit to prevent an attach event before the USB OTG controller has properly initialized. Clearing this bit causes a detach event.

10.3.4.2 USB Status Register (USBSTS)

This register indicates various states of each module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them.

Address: 0xFC0B_0144 (USBSTS)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	TI1	TI0	0	0	0	0	UPI	UAI	0	NAKI
W							w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH	0	ULPII	0		SRI	URI	AAI	SEI	FRI	PCI	UEI	UI
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 10-18. USB Status Register (USBSTS)

Table 10-20. USBSTS Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25 TI1	General purpose timer 1 interrupt. Set when the counter in the GPTIMER1CTRL register transitions to zero. Writing a one to this bit clears it. 0 No interrupt 1 Interrupt occurred.
24 TI0	General purpose timer 0 interrupt. Set when the counter in the GPTIMER0CTRL register transitions to zero. Writing a one to this bit clears it. 0 No interrupt 1 Interrupt occurred.
23–20	Reserved, must be cleared.
19 UPI	USB host periodic interrupt. Set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the periodic schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the periodic schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. **Note:** This bit is not used by the device controller and is always zero.
18 UAI	USB host asynchronous interrupt. Set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the asynchronous schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the asynchronous schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. **Note:** This bit is not used by the device controller and is always zero.
17	Reserved, must be cleared.
16 NAKI	NAK interrupt. Set by hardware for a particular endpoint when the TX/RX endpoint's NAK bit and the corresponding TX/RX endpoint's NAK enable bit are set. The hardware automatically clears this bit when all the enabled TX/RX endpoint NAK bits are cleared.
15 AS	Asynchronous schedule status. Reports the current real status of asynchronous schedule. Controller is not immediately required to disable or enable the asynchronous schedule when software transitions the USBCMD[ASE] bit. When this bit and the USBCMD[ASE] bit have the same value, the asynchronous schedule is enabled (1) or disabled (0). Used only in host mode. 0 Disabled. 1 Enabled.
14 PS	Periodic schedule status. Reports current real status of periodic schedule. Controller is not immediately required to disable or enable the periodic schedule when software transitions the USBCMD[PSE] bit. When this bit and the USBCMD[PSE] bit have the same value, the periodic schedule is enabled or disabled. Used only in host mode. 0 Disabled. 1 Enabled.
13 RCL	Reclamation. DetectS an empty asynchronous schedule. Used only by the host mode. 0 Non-empty asynchronous schedule. 1 Empty asynchronous schedule.

Table 10-20. USBSTS Field Descriptions (continued)

Field	Description
12 HCH	Host controller halted. This bit is cleared when the USBCMD[RS] bit is set. The controller sets this bit after it stops executing because of the USBCMD[RS] bit being cleared, by software or the host controller hardware (for example, internal error). Used only in host mode. 0 Running. 1 Halted.
11	Reserved, must be cleared.
10 ULPII	ULPI interrupt. Set by event completion.
9	Reserved, must be cleared.
8	Device-controller suspend. Non-EHCI bit. When a device controller enters a suspend state from an active state, this bit is set. The device controller clears the bit upon exiting from a suspend state. Used only by the device controller. 0 Active. 1 Suspended.
7 SRI	SOF received. This is a non-EHCI status bit. Software writes a 1 to this bit to clear it. Host mode: In host mode, this bit is set every 125 μs, provided PHY clock is present and running (for example, the port is NOT suspended) and can be used by the host-controller driver as a time base. Device mode: When controller detects a start of (micro) frame, bit is set. When a SOF is extremely late, controller automatically sets this bit to indicate an SOF was expected. Therefore, this bit is set roughly every 1 ms in device FS mode and every 125 μsec in HS mode, and it is synchronized to the actual SOF received. Because the controller is initialized to FS before connect, this bit is set at an interval of 1 ms during the prelude to the connect and chirp.
6 URI	USB reset received. A non-EHCI bit. When the controller detects a USB reset and enters the default state, this bit is set. Software can write a 1 to this bit to clear it. Used only by in device mode. 0 No reset received. 1 Reset received.
5 AAI	Interrupt on async advance. By setting the USBCMD[IAA] bit, system software can force the controller to issue an interrupt the next time the controller advances the asynchronous schedule. This status bit indicates the assertion of that interrupt source. Used only by the host mode. 0 No async advance interrupt. 1 Async advance interrupt.
4 SEI	System error. Set when an error is detected on the system bus. If the system error enable bit (USBINTR[SEE]) is set, interrupt generates. The interrupt and status bits remain set until cleared by writing a 1 to this bit. Additionally, when in host mode, the USBCMD[RS] bit is cleared, effectively disabling controller. An interrupt generates for the USB OTG controller in device mode, but no other action is taken. 0 Normal operation 1 Error
3 FRI	Frame-list rollover. Controller sets this bit when the frame list index (FRINDEX) rolls over from its maximum value to 0. The exact value the rollover occurs depends on the frame list size. For example, if the frame list size (as programmed in the USBCMD[FS] field) is 1024, the frame index register rolls over every time FRINDEX[13] toggles. Similarly, if the size is 512, the controller sets this bit each time FRINDEX[12] toggles. Used only in the host mode.

Table 10-20. USBSTS Field Descriptions (continued)

Field	Description
2 PCI	Port change detect. This bit is not EHCI compatible. Host mode: Controller sets this bit when a connect status occurs on any port, a port enable/disable change occurs, an over-current change occurs, or the force port resume (PORTSC*n*[FPR]) bit is set as the result of a J-K transition on the suspended port. Device mode: The controller sets this bit when it enters the full- or high-speed operational state. When it exits the full- or high-speed operation states due to reset or suspend events, the notification mechanisms are URI and bits respectively. The device controller detects resume signaling only.
1 UEI	USB error interrupt. When completion of USB transaction results in error condition, the controller sets this bit. If the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set, this bit is set along with the USBINT bit. See Section 4.15.1 in the EHCI specification for a complete list of host error interrupt conditions. See Table 10-58 for more information on device error matrix. 0 No error. 1 Error detected.
0 UI	USB interrupt (USBINT). This bit is set by the controller when the cause of an interrupt is a completion of a USB transaction where the TD has an interrupt on complete (IOC) bit set. This bit is also set by the controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

10.3.4.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt generates when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) continues to show interrupt sources (even if the USBINTR register disables them), allowing polling of interrupt events by the software.

Address: 0xFC0B_0148 (USBINTR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	TIE1	TIE0	0	0	0	0	UPIE	UAIE	0	NAKE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	ULPIE	0	SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-19. USB Interrupt Enable Register (USBINTR)

Table 10-21. USBINTR Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25 TIE1	General purpose timer 1 interrupt enable. When this bit and USBSTS[GPTINT1] are set, the USB controller issues an interrupt to the processor. The interrupt is acknowledged by clearing GPTINT1. 0 Disabled 1 Enabled
24 TIE0	General purpose timer 0 interrupt enable. When this bit and USBSTS[GPTINT0] are set, the USB controller issues an interrupt to the processor. The interrupt is acknowledged by clearing GPTINT0. 0 Disabled 1 Enabled
23–20	Reserved, must be cleared.
19 UPIE	USB host periodic interrupt enable. When this bit and USBSTS[USBHSTPERINT] are set, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing USBHSTPERINT.
18 UAIE	USB host asynchronous interrupt enable. When this bit and USBSTS[USBHSTASYNCINT] are set, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing USBHSTASYNCINT.
17	Reserved, must be cleared.
16 NAKE	NAK interrupt enable. When this bit and the USBSTS[NAKI] bit are set, an interrupt generates. 0 Disabled 1 Enabled
15–11	Reserved, must be cleared.
10 ULPIE	ULPI enable. When this bit and USBSTS[ULPII] are set, controller issues an interrupt. The interrupt is acknowledged by writing a 1 to USBSTS[ULPII].
9	Reserved, must be cleared.
8 SLE	Sleep (DC suspend) enable. A non-EHCI bit. When this bit is set and the USBSTS[] bit transitions, USB OTG controller issues an interrupt. Software writing a 1 to the USBSTS[] bit acknowledges the interrupt. Used only in device mode. 0 Disabled 1 Enabled
7 SRE	SOF-received enable. This is a non-EHCI bit. When this bit and the USBSTS[SRI] bit are set, controller issues an interrupt. Software clearing the USBSTS[SRI] bit acknowledges the interrupt. 0 Disabled 1 Enabled
6 URE	USB-reset enable. A non-EHCI bit. When this bit and the USBSTS[URI] bit are set, device controller issues an interrupt. Software clearing the USBSTS[URI] bit acknowledges the interrupt. Used only in device mode. 0 Disabled 1 Enabled
5 AAE	Interrupt on async advance enable. When this bit and the USBSTS[AAI] bit are set, controller issues an interrupt at the next interrupt threshold. Software clearing the USBSTS[AAI] bit acknowledges the interrupt. Used only in host mode. 0 Disabled 1 Enabled

Table 10-21. USBINTR Field Descriptions (continued)

Field	Description
4 SEE	System error enable. When this bit and the USBSTS[SEI] bit are set, controller issues an interrupt. Software clearing the USBSTS[SEI] bit acknowledges the interrupt. 0 Disabled 1 Enabled
3 FRE	Frame list rollover enable. When this bit and the USBSTS[FRI] bit are set, controller issues an interrupt. Software clearing the USBSTS[FRI] bit acknowledges the interrupt. Used only in host mode. 0 Disabled 1 Enabled
2 PCE	Port change detect enable. When this bit and the USBSTS[PCI] bit are set, controller issues an interrupt. Software clearing the USBSTS[PCI] bit acknowledges the interrupt. 0 Disabled 1 Enabled
1 UEE	USB error interrupt enable. When this bit and the USBSTS[UEI] bit are set, controller issues an interrupt at the next interrupt threshold. Software clearing the USBSTS[UEI] bit acknowledges the interrupt. 0 Disabled 1 Enabled
0 UE	USB interrupt enable. When this bit is 1 and the USBSTS[UI] bit is set, the USB OTG controller issues an interrupt at the next interrupt threshold. Software clearing the USBSTS[UI] bit acknowledges the interrupt. 0 Disabled 1 Enabled

10.3.4.4 Frame Index Register (FRINDEX)

In host mode, the controller uses this register to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits [N–3] select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the USBCMD[FS] field.

This register must be a longword. Byte writes produce undefined results. This register cannot be written unless the USB OTG controller is in halted state as the USBSTS[HCH] bit indicates. A write to this register while the USBSTS[RS] bit is set produces undefined results. Writes to this register also affect the SOF value.

In device mode, this register is read-only, and the USB OTG controller updates the FRINDEX[13–3] bits from the frame number the SOF marker indicates. When the USB bus receives a SOF, FRINDEX[13–3] checks against the SOF marker. If FRINDEX[13–3] is different from the SOF marker, FRINDEX[13–3] is set to the SOF value and FRINDEX[2–0] is cleared (SOF for 1 ms frame). If FRINDEX[13–3] equals the SOF value, FRINDEX[2–0] is incremented (SOF for 125 µsec microframe.)

Address: 0xFC0B_014C (FRINDEX) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRINDEX													
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-20. Frame Index Register (FRINDEX)

Table 10-22. FRINDEX Field Descriptions

Field	Description
31–14	Reserved, must be cleared.
13–0 FRINDEX	Frame index. The value in this register increments at the end of each time frame (microframe). Bits [N– 3] are for the frame list current index. This means each location of the frame list is accessed 8 times per frame (once each microframe) before moving to the next index. In device mode, the value is the current frame number of the last frame transmitted and not used as an index. In either mode, bits 2–0 indicate current microframe.

Table 10-23 illustrates values of N based on the value of the USBCMD[FS] field when used in host mode.

Table 10-23. FRINDEX N Values

USBCMD[FS]	Frame List Size	FRINDEX N value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

10.3.4.5 Periodic Frame List Base Address Register (PERIODICLISTBASE)

This register contains the beginning address of the periodic frame list in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer assumes to be 4-Kbyte aligned. The contents combine with the FRINDEX register to enable the controller to step through the periodic frame list in sequence.

The host and device mode functions share this register. In host mode, it is the PERIODICLISTBASE register; in device mode, it is the DEVICEADDR register. See Section 10.3.4.6, “Device Address Register (DEVICEADDR),” for more information.

Address: 0xFC0B_0154 (PERIODICLISTBASE)　　Access: User 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
R	PERBASE																				0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-21. Periodic Frame List Base Address Register (PERIODICLISTBASE)

Table 10-24. PERIODICLISTBASE Field Descriptions

Field	Description
31–12 PERBASE	Base Address. These bits correspond to memory address signal [31:12]. Used only in the host mode
11–0	Reserved, must be cleared.

10.3.4.6 Device Address Register (DEVICEADDR)

This register is not defined in the EHCI specification. For device mode, the upper seven bits of this register represent the device address. After any controller or USB reset, the device address is set to the default address (0). The default address matches all incoming addresses. Software reprograms the address after receiving a SET_ADDRESS descriptor.

The host and device mode functions share this register. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See Section 10.3.4.5, “Periodic Frame List Base Address Register (PERIODICLISTBASE),” for more information.

Address: 0xFC0B_0154 (DEVICEADDR) Access: User 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
R	USBADR							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-22. Device Address Register (DEVICEADDR)

Table 10-25. DEVICEADDR Field Descriptions

Field	Description
31–25 USBADR	Device Address. This field corresponds to the USB device address.
24–0	Reserved, must be cleared.

10.3.4.7 Current Asynchronous List Address Register (ASYNCLISTADDR)

The ASYNCLISTADDR register contains the address of the next asynchronous queue head to executed by the host.

The host and device mode functions share this register. In host mode, it is the ASYNCLISTADDR register; in device mode, it is the EPLISTADDR register. See Section 10.3.4.8, “Endpoint List Address Register (EPLISTADDR),” for more information.

Address: 0xFC0B_0158 (ASYNCLISTADDR) Access: User 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
R	ASYBASE																											0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-23. Current Asynchronous List Address Register (ASYNCLISTADDR)

Table 10-26. ASYNCLISTADDR Field Descriptions

Field	Description
31–5 ASYBASE	Link pointer low (LPL). These bits correspond to memory address signal [31:5]. This field may only reference a queue head (QH). Used only in host mode.
4–0	Reserved, must be cleared.

10.3.4.8 Endpoint List Address Register (EPLISTADDR)

This register is not defined in the EHCI specification. For device mode, this register contains the address of the endpoint list top in system memory. The memory structure referenced by this physical memory pointer assumes to be 64-bytes. The queue head is actually a 48-byte structure, but must be aligned on 64-byte boundary. However, the EPBASE field has a granularity of 2 Kbytes; in practice, the queue head should be 2-Kbyte aligned.

The host and device mode functions share this register. In device mode, it is the EPLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See Section 10.3.4.7, “Current Asynchronous List Address Register (ASYNCLISTADDR),” for more information.

Address: 0xFC0B_0158 (EPLISTADDR) Access: User 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
R	EPBASE																					0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-24. Endpoint List Address Register (EPLISTADDR)

Table 10-27. EPLISTADDR Field Descriptions

Field	Description
31–11 EPBASE	Endpoint list address. Correspond to memory address signals [31:11] References a list of up to 32 queue heads (i.e. one queue head per endpoint and direction). Address of the top of the endpoint list.
10–0	Reserved, must be cleared.

10.3.4.9 Host TT Asynchronous Buffer Control (TTCTRL)

Address: 0xFC0B_015C (TTCTRL) Access: User 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
R	0	TTHA							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-25. Host TT Asynchronous Buffer Control (TTCTRL)

Table 10-28. TTCTRL Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–24 TTHA	TT Hub Address. This field is used to match against the Hub Address field in a QH or siTD to determine if the packet is routed to the internal TT for directly attached FS/LS devices. If the hub address in the QH or siTD does not match this address then the packet is broadcast on the high speed ports destined for a downstream HS hub with the address in the QH or siTD.
23–0	Reserved, must be cleared.

10.3.4.10 Master Interface Data Burst Size Register (BURSTSIZE)

This register is not defined in the EHCI specification. BURSTSIZE dynamically controls the burst size during data movement on the initiator (master) interface.

Address: 0xFC0B_0160 (BURSTSIZE) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TXPBURST								RXPBURST							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 10-26. Master Interface Data Burst Size (BURSTSIZE)

Table 10-29. BURSTSIZE Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 TXPBURST	Programable TX burst length. Represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater than 16.
7–0 RXPBURST	Programable RX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16.

10.3.4.11 Transmit FIFO Tuning Control Register (TXFILLTUNING)

This register is not defined in the EHCI specification. The TXFILLTUNING register controls performance tuning associated with how the module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include how much data to post into the FIFO and an estimate for how long that operation takes in the target system.

Definitions:

T_0 = Standard packet overhead

T_1 = Time to send data payload

T_s = Total packet flight time (send-only) packet ($T_s = T_0 + T_1$)

T_{ff} = Time to fetch packet into TX FIFO up to specified level

T_p = Total packet time (fetch and send) packet ($T_p = T_{ff} + T_s$)

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, the host controller checks to ensure T_p remains before the end of the (micro)frame. If so, it pre-fills the TX FIFO. If at anytime during the pre-fill operation the time remaining the (micro)frame is less than T_s, packet attempt ceases and tries at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to mark the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic beginning after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and should be minimized (not necessarily eliminated). The TSCHHEALTH (T_{ff}) parameter described below can minimize back-offs.

Address: 0xFC0B_0164 (TXFILLTUNING) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	TXFIFOTHRES						0	0	0	TXSCHHEALTH					TXSCHOH							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-27. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 10-30. TXFILLTUNING Field Descriptions

Field	Description
31–22	Reserved, must be cleared.
21–16 TXFIFOTHRES	FIFO burst threshold. Controls the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on the bus. The minimum value is 2 and this value should be as low as possible to maximize USB performance. Systems with unpredictable latency and/or insufficient bandwidth can use a higher value where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can replenish from system memory. This value is ignored if the USBMODE[SDIS] bit is set. When the USBMODE[SDIS] bit is set, the host controller behaves as if TXFIFOTHRES is set to its maximum value.
15–13	Reserved, must be cleared.

Table 10-30. TXFILLTUNING Field Descriptions (continued)

Field	Description
12–8 TXSCHHEALTH	Scheduler health counter. These bits increment when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next SOF. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter stops counting after reaching the maximum of 31.
7–0 TXSCHOH	Scheduler overhead. These bits add an additional fixed offset to the schedule time estimator described as T_{ff}. As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH field to less than 10 per second in a highly utilized bus. Choosing a value too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267 µs when a device connects in high-speed mode. The time unit represented in this register is 6.333 µs when a device connects in low-/full-speed mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: $\frac{\text{TXFIFOTHRES} \times (\text{BURSTSIZE} \times 4)}{40 \times \mathit{TimeUnit}}$ ***Eqn. 10-1*** Always rounded to the next higher integer. *TimeUnit* is 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, set TXSCHOH to 5×(8×4)/(40×1.267) equals 4 for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, low the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, raise the value by 1. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.

10.3.4.12 ULPI Register Access (ULPI_VIEWPORT)

The register provides indirect access to the ULPI PHY register set. Although the controller modules perform access to the ULPI PHY register set, there may be circumstances where software may need direct access.

NOTE

Be advised that writes to the ULPI through the ULPI viewport can substantially harm standard USB operations. Currently no usage model has been defined where software should need to execute writes directly to the ULPI. Executing read operations though the ULPI viewport should have no harmful side effects to standard USB operations. Also, if the ULPI interface is not enabled, this register is always read cleared.

Address: 0xFC0B_0170 (ULPI_VIEWPORT) Access: User 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
R	ULPI_WU	ULPI_RUN	ULPI_RW	0	ULPI_SS	ULPI_PORT			ULPI_ADDR								ULPI_DATRD								ULPI_DATWR							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-28. ULPI Register Access (ULPI VIEWPORT)

Table 10-31. ULPI VIEWPORT Field Descriptions

Field	Description
31 ULPI_WU	ULPI wake-up. Setting this bit begins the wake-up operation. This bit automatically clears after the wake-up is complete. After this bit is set, it can not be cleared by software. **Note:** The driver must never execute a wake-up and a read/write operation at the same time.
30 ULPI_RUN	ULPI run. Setting this bit begins a read/write operation. This bit automatically clears after the read/write is complete. After this bit is set, it can not be cleared by software. **Note:** The driver must never execute a wake-up and a read/write operation at the same time.
29 ULPI_RW	Read/write. Selects between running a read or write operation to the ULPI. 0 Read 1 Write
28	Reserved, should be cleared.
27 ULPI_SS	Sync state. Represents the state of the ULPI interface. Before reading this bit, the ULPI_PORT field should be set accordingly if used with the multi-port host. Otherwise, this field should always remain 0. 0 Any other state (that is, carkit, serial, low power). 1 Normal sync state.
26–24 ULPI_PORT	Port number. For wake-up or read/write operations this value selects the port number to which the ULPI PHY is attached. Valid values are 0 and 1.
23–16 ULPI_ADDR	Data address. When a read or write operation is commanded, the address of the operation is written to this field.
15–8 ULPI_DATRD	Data read. After a read operation completes, the result is placed in this field.
7–0 ULPI_DATWR	Data write. When a write operation is commanded, the data to be sent is written to this field.

There are two operations that can be performed with the ULPI viewport, wake-up and read/write operations. The wake-up operation is used to put the ULPI interface into normal operation mode and re-enable the clock if necessary. A wake-up operation is required before accessing the registers when the ULPI interface is operating in low power mode, serial mode, or carkit mode. The ULPI state can be determined by reading the sync state bit (ULPI_SS). If this bit is set, then the ULPI interface is running in normal operating mode and can accept read/write operations. If ULPI_SS is cleared, then read/write operations are not executed. Undefined behavior results if a read or write operation is performed when ULPI_SS is cleared. To execute a wake-up operation, write all 32-bits of the ULPI VIEWPORT where ULPI_PORT is constructed appropriately and the ULPI_WU bit is set and the ULPI_RUN bit is cleared. Poll the ULPI VIEWPORT until ULPI_WU is cleared for the operation to complete.

To execute a read or write operation, write all 32-bits of the ULPI VIEWPORT where ULPI_DATWR, ULPI_ADDR, ULPI_PORT, ULPI_RW are constructed appropriately and the ULPI_RUN bit is set. Poll the ULPI VIEWPORT until ULPI_RUN is cleared for the operation to complete. For read operations, ULPI_DATRD is valid after ULPI_RUN is cleared.

The polling method above can be replaced with interrupts using the ULPI interrupt defined in the USBSTS and USBINTR registers. When a wake-up or read/write operation completes, the ULPI interrupt is set.

10.3.4.13 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of a 0x0000_0001 to indicate that all port routings default to this host controller.

Address: 0xFC0B_0180 (CONFIGFLAG) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 10-29. Configure Flag Register (CONFIGFLAG)

Table 10-32. CONFIGFLAG Field Descriptions

Field	Description
31–0	Reserved. (0x0000_0001, all port routings default to this host)

10.3.4.14 Port Status and Control Registers (PORTSC*n*)

The USB module contains a single PORTSC register. This register only resets when power is initially applied or in response to a controller reset. Initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power to one.

For the USB OTG module in device mode, the USB OTG controller does not support power control. Port control in device mode is used only for status port reset, suspend, and current connect status. It is also used to initiate test mode or force signaling, and allows software to place the PHY into low-power suspend mode and disable the PHY clock.

Address: 0xFC0B_0184 (PORTSC1) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PTS		1	0	PSPD		0	PFSC	PHCD	WKOC	WKDS	WLCN	PTC			
W																
Reset	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		PO	PP	LS		HSP	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS
W											w1c		w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 10-30. Port Status and Control Register (PORTSC1*)*

Table 10-33. PORTSC1 Field Descriptions

Field	Description
31–30 PTS	Port transceiver select. Controls which parallel transceiver interface is selected. 00 Reserved 01 Reserved 10 ULPI parallel interface 11 FS/LS on-chip transceiver This bit is not defined in the EHCI specification.
29	Reserved, must be set.
28	Reserved, must be cleared.
27–26 PSPD	Port speed. This read-only register field indicates the speed the port operates. This bit is not defined in the EHCI specification. 00 Full speed 01 Low speed 10 High speed 11 Undefined
25	Reserved, must be cleared.
24 PFSC	Port force full-speed connect. Disables the chirp sequence that allows the port to identify itself as a HS port. useful for testing FS configurations with a HS host, hub, or device. Not defined in the EHCI specification. 0 Allow the port to identify itself as high speed. 1 Force the port to only connect at full speed. This bit is for debugging purposes.
23 PHCD	PHY low power suspend. This bit is not defined in the EHCI specification. Host mode: The PHY can be placed into low-power suspend when downstream device is put into suspend mode or when no downstream device connects. Software completely controls low-power suspend. Device mode: For the USB OTG module in device mode, the PHY can be put into low power suspend when the device is not running (USBCMD[RS] = 0) or suspend signaling is detected on the USB. The PHCD bit is cleared automatically when the resume signaling is detected or when forcing port resumes. 0 Normal PHY operation. 1 Signal the PHY to enter low-power suspend mode Reading this bit indicates the status of the PHY.
22 WKOC	Wake on over-current enable. Enables the port to be sensitive to over-current conditions as wake-up events. This field is 0 if the PP bit is cleared. In host mode, this bit can work with an external power control circuit.
21 WKDS	Wake on disconnect enable. Enables the port to be sensitive to device disconnects as wake-up events. This field is 0 if the PP bit is cleared or the module is in device mode. In host mode, this bit can work with an external power control circuit.
20 WLCN	Wake on connect enable. Enables the port to be sensitive to device connects as wake-up events. This field is 0 if the PP bit is cleared or the module is in device mode. In host mode, this can work with an external power control circuit.

Table 10-33. PORTSC1 Field Descriptions (continued)

Field	Description
19–16 PTC	Port test control. Any value other than 0 indicates the port operates in test mode. Refer to Chapter 7 of the *USB Specification Revision 2.0* for details on each test mode. 0000 Not enabled. 0001 J_STATE 0010 K_STATE 0011 SEQ_NAK 0100 Packet 0101 FORCE_ENABLE_HS 0110 FORCE_ENABLE_FS 0111 FORCE_ENABLE_LS Else Reserved. **Note:** The FORCE_ENABLE_FS and FORCE ENABLE_LS settings are extensions to the test mode support in the EHCI specification. Writing the PTC field to any of the FORCE_ENABLE values forces the port into the connected and enabled state at the selected speed. Then clearing the PTC field allows the port state machines to progress normally from that point.
15–14 PIC	Port indicator control.For this device, this feature is not implemented, therefore this field is read-only and is always cleared.
13 PO	Port owner. Port owner handoff is not implemented in this design, therefore this bit is read-only and is always cleared.
12 PP	Port power. Represents the current setting of the port power control switch (0 equals off, 1 equals on). When power is not available on a port (PP = 0), it is non-functional and does not report attaches, detaches, etc. When an over-current condition is detected on a powered port, the host controller driver from a 1to a 0 (removing power from the port) transitions the PP bit in each affected port.
11–10 LS	Line status. Reflects current logical levels of the USB DP (bit 11) and DM (bit 10) signal lines. In host mode, the line status by the host controller driver is not necessary (unlike EHCI) because hardware manages the connection of FS and LS. In device mode, LS by the device controller is not necessary. 00 SE0 01 J-state 10 K-state 11 Undefined
9 HSP	High speed port. Indicates if the host/device connected is in high speed mode. 0 FS or LS 1 HS **Note:** This bit is redundant with the PSPD bit field.
8 PR	Port reset. This field is cleared if the PP bit is cleared. Host mode: When software sets this bit the bus-reset sequence as defined in the *USB Specification Revision 2.0* starts. This bit automatically clears after the reset sequence completes. This behavior is different from EHCI where the host controller driver is required to clear this bit after the reset duration is timed in the driver. Device mode: This bit is a read-only status bit. Device reset from the USB bus is also indicated in the USBSTS register. 0 Port is not in reset. 1 Port is in reset.

Table 10-33. PORTSC1 Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7
SUSP</td><td>Suspend
0 Port not in suspend state.
1 Port in suspend state.
Host mode:
The PE and SUSP bits define the port state as follows:
<table>
<tr><th>PE</th><th>SUSP</th><th>Port State</th></tr>
<tr><td>0</td><td>x</td><td>Disable</td></tr>
<tr><td>1</td><td>0</td><td>Enable</td></tr>
<tr><td>1</td><td>1</td><td>Suspend</td></tr>
</table>
When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was set. In the suspend state, the port is sensitive to resume detection. The bit status does not change until the port is suspended and there may be a delay in suspending a port if there is a transaction currently in progress on the USB.
The module unconditionally clears this bit when software clears the FPR bit. The host controller ignores clearing this bit. If host software sets this bit when the port is not enabled (PE = 0), the results are undefined.
This field is cleared if the PP bit is cleared in host mode.
Device mode:
In device mode, this bit is a read-only status bit.</td></tr>
<tr><td>6
FPR</td><td>Force Port Resume. This bit is not-EHCI compatible.
0 No resume (K-state) detected/driven on port.
1 Resume detected/driven on port.
Host mode:
Software sets this bit to drive resume signaling. The controller sets this bit if a J-to-K transition is detected while the port is in suspend state (PE = SUSP = 1), which in turn sets the USBSTS[PCI] bit. This bit automatically clears after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to clear this bit after the resume duration is timed in the driver.
When the controller owns the port, the resume sequence follows the defined sequence documented in the <i>USB Specification Revision 2.0</i>. The resume signaling (full-speed K) is driven on the port as long as this bit remains set. This bit remains set until the port switches to the high-speed idle. Clearing this bit has no affect because the port controller times the resume operation to clear the bit the port control state switches to HS or FS idle.
This field is cleared if the PP bit is cleared in host mode.
Device mode:
After the device is in suspend state for 5 ms or more, software must set this bit to drive resume signaling before clearing. The device controller sets this bit if a J-to-K transition is detected while port is in suspend state, which in turn sets the USBSTS[PCI] bit. The bit is cleared when the device returns to normal operation.</td></tr>
<tr><td>5
OCC</td><td>Over-current change. Indicates a change to the OCA bit. Software clears this bit by writing a 1. For host mode, the user can provide over-current detection to the USB<i>n</i>_PWRFAULT signal for this condition. For device-only implementations, this bit must always be cleared.
0 No over-current.
1 Over-current detect.</td></tr>
<tr><td>4
OCA</td><td>Over-current active. This bit automatically transitions from 1 to 0 when the over-current condition is removed. For host/OTG implementations, the user can provide over-current detection to the USB<i>n</i>_PWRFAULT signal for this condition. For device-only implementations, this bit must always be cleared.
0 Port not in over-current condition.
1 Port currently in over-current condition.</td></tr>
</table>

Table 10-33. PORTSC1 Field Descriptions (continued)

Field	Description
3 PEC	Port enable/disable change. For the root hub, this bit gets set only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the *USB Specification*). Software clears this by writing a 1 to it. In device mode, the device port is always enabled. (This bit is zero). 0 No change. 1 Port disabled. This field is cleared if the PP bit is cleared.
2 PE	Port enabled/disabled. Host mode: Ports can only be enabled by the controller as a part of the reset and enable sequence. Software cannot enable a port by setting this bit. A fault condition (disconnect event or other fault condition) or host software can disable ports. The bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host and bus events. When the port is disabled, downstream propagation of data is blocked except for reset. This field is cleared if the PP bit is cleared in host mode. Device mode: The device port is always enabled. (This bit is set).
1 CSC	Connect change status. Host mode: This bit indicates a change occurred in the port's current connect status. The controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition; hub hardware is setting an already-set bit (i.e., the bit remains set). Software clears this bit by writing a 1 to it. This field is cleared if the PP bit is cleared. 0 No change. 1 Connect status has changed. In device mode, this bit is undefined.
0 CCS	Current connect status. Indicates that a device successfully attaches and operates in high speed or full speed as indicated by the PSPD bit. If clear, the device did not attach successfully or forcibly disconnects by the software clearing the USBCMD[RUN] bit. It does not state the device disconnected or suspended. This field is cleared if the PP bit is cleared in host mode. 0 No device present (host mode) or attached (device mode) 1 Device is present (host mode) or attached (device mode)

10.3.4.15 On-the-Go Status and Control Register (OTGSC)

This register is not defined in the EHCI specification. The host controller implements one OTGSC register corresponding to port 0 of the host controller.

The OTGSC register has four sections:

- OTG interrupt enables (read/write)
- OTG interrupt status (read/write to clear)
- OTG status inputs (read-only)
- OTG controls (read/write)

The status inputs de-bounce using a 1 ms time constant. Values on the status inputs that do not persist for more than 1 ms do not cause an update of the status inputs or an OTG interrupt.

Address: 0xFC0B_01A4 (OTGSC) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	DPIE	1MSE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	0	DPIS	1MSS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS
W										w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	DPS	1MST	BSE	BSV	ASV	AVV	ID	0	0	IDPU	DP	OT	0	VC	VD
W																
Reset	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Figure 10-31. On-the-Go Status and Control Register (OTGSC)

Table 10-34. OTGSC Field Descriptions

Field	Description
31	Reserved, must be cleared.
30 DPIE	Data pulse interrupt enable. 0 Disable 1 Enable
29 1MSE	1 millisecond timer interrupt enable. 0 Disable 1 Enable
28 BSEIE	B session end interrupt enable. 0 Disable 1 Enable
27 BSVIE	B session valid interrupt enable. 0 Disable 1 Enable
26 ASVIE	A session valid interrupt enable. 0 Disable 1 Enable
25 AVVIE	A VBUS valid interrupt enable. 0 Disable 1 Enable
24 IDIE	USB ID interrupt enable. 0 Disable 1 Enable
23	Reserved, must be cleared.
22 DPIS	Data pulse interrupt status. Indicates when data bus pulsing occurs on DP or DM. Data bus pulsing only detected when USBMODE[CM] equals 11 and PORTSC0[PP] is cleared. Software must write a 1 to clear this bit.
21 1MSS	1 millisecond timer interrupt status. This bit is set once every millisecond. Software must write a 1 to clear this bit.
20 BSEIS	B session end interrupt status. Indicates when VBUS falls below the B session end threshold. Software must write a 1 to clear this bit.

Table 10-34. OTGSC Field Descriptions (continued)

Field	Description
19 BSVIS	B session valid interrupt status. Indicates when VBUS rises above or falls below the B session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
18 ASVIS	A session valid interrupt status. Indicates when VBUS rises above or falls below the A session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
17 AVVIS	A VBUS valid interrupt status. Indicates when VBUS rises above or falls below the VBUS valid threshold (4.4 VDC) on an A device. Software must write a 1 to clear this bit.
16 IDIS	USB ID interrupt status. Indicates when a change on the ID input is detected. Software must write a 1 to clear this bit.
15	Reserved, must be cleared.
14 DPS	Data bus pulsing status. 0 No pulsing on port. 1 Pulsing detected on port.
13 1MST	1 millisecond timer toggle. This bit toggles once per millisecond.
12 BSE	B session end. 0 VBus is above B session end threshold. 1 VBus is below B session end threshold.
11 BSV	B Session valid. 0 VBus is below B session valid threshold. 1 VBus is above B session valid threshold.
10 ASV	A Session valid. 0 VBus is below A session valid threshold. 1 VBus is above A session valid threshold.
9 AVV	A VBus valid. 0 VBus is below A VBus valid threshold. 1 VBus is above A VBus valid threshold.
8 ID	USB ID. 0 A device. 1 B device.
7–6	Reserved, must be cleared.
5 IDPU	ID Pull-up. Provides control over the ID pull-up resistor. 0 Disable pull-up. ID input not sampled. 1 Enable pull-up.
4 DP	Data pulsing. 0 The pull-up on DP is not asserted. 1 The pull-up on DP is asserted for data pulsing during SRP.
3 OT	OTG Termination. This bit must be set with the OTG module in device mode. 0 Disable pull-down on DM. 1 Enable pull-down on DM.
2	Reserved, must be cleared.

Table 10-34. OTGSC Field Descriptions (continued)

Field	Description
1 VC	VBUS charge. Setting this bit causes the VBUS line to charge. This is used for VBus pulsing during SRP.
0 VD	VBUS discharge. Setting this bit causes VBUS to discharge through a resistor.

10.3.4.16 USB Mode Register (USBMODE)

This register is not defined in the EHCI specification. It controls the operating mode of the module.

Address: 0xFC0B_01A8 (USBMODE) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	SDIS	SLOM	ES	CM	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-32. USB Mode Register (USBMODE)

Table 10-35. USBMODE Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 SDIS	Stream disable. 0 Inactive. 1 Active. Host mode: Setting this bit ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency fills to capacity before the packet launches onto the USB. Time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature. Also, in systems with high system bus utilization, setting this bit ensures no overruns or underruns during operation at the expense of link utilization. SDIS can be left clear and the rules under the description of the TXFILLTUNING register can limit underruns/overruns for those who desire optimal link performance. Device mode: Setting this bit disables double priming on RX and TX for low bandwidth systems. This mode ensures that when the RX and TX buffers are sufficient to contain an entire packet that the standard double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems. In high-speed mode, all packets received are responded to with a NYET handshake when stream disable is active.
3 SLOM	Setup lockout mode. For the module in device mode, this bit controls behavior of the setup lock mechanism. See Section 10.5.3.4.4, “Control Endpoint Operation.” 0 Setup lockouts on. 1 Setup lockouts off (software requires use of the USBCMD[SUTW] bit).

Table 10-35. USBMODE Field Descriptions (continued)

Field	Description
2 ES	Endian select. Controls the byte ordering of the transfer buffers to match the host microprocessor bus architecture. The bit fields in the register interface and the DMA data structures (including the setup buffer within the device QH) are unaffected by the value of this bit, because they are based upon 32-bit words. 0 Little endian. First byte referenced in least significant byte of 32-bit word. 1 Big endian. First byte referenced in most significant byte of 32-bit word. **Note:** For proper operation, this bit must be set for this ColdFire device.
1–0 CM	Controller mode. This register can be written only once after reset. If necessary to switch modes, software must reset the controller by writing to the USBCMD[RST] bit before reprogramming this register. 00 Idle (default for the USB OTG module) 01 Reserved 10 Device controller 11 Host controller **Note:** The USB OTG module must be initialized to the desired operating mode after reset.

10.3.4.17 Endpoint Setup Status Register (EPSETUPSR)

This register is not defined in the EHCI specification. This register contains the endpoint setup status and is used only in device mode.

Address: 0xFC0B_01AC (EPSETUPSR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EPSETUPSTAT			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-33. Endpoint Setup Status Register (EPSETUPSR)

Table 10-36. EPSETUPSR Field Descriptions

Field	Description
31–4	Reserved, must be cleared.
3–0 EPSETUPSTAT	Setup endpoint status. For every setup transaction received, a corresponding bit in this field is set. Software must clear or acknowledge the setup transfer by writing a 1 to a respective bit after it has read the setup data from the queue head. The response to a setup packet, as in the order of operations and total response time, is crucial to limit bus time outs while the setup lockout mechanism engages.

10.3.4.18 Endpoint Initialization Register (EPPRIME)

This register is not defined in the EHCI specification. This register is used to initialize endpoints and is used only in device mode.

Address: 0xFC0B_01B0 (EPPRIME) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	PETB				0	0	0	0	0	0	0	0	0	0	0	0	PERB			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-34. Endpoint Initialization Register (EPPRIME)

Table 10-37. EPPRIME Field Descriptions

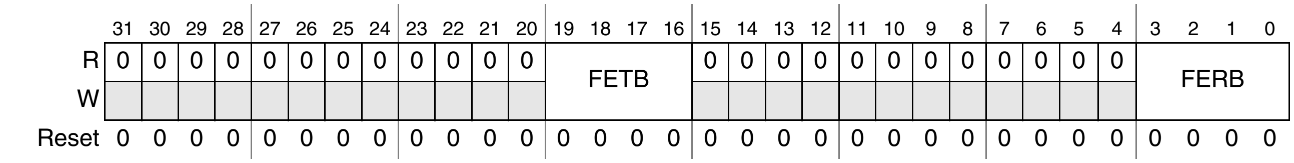
Field	Description
31–20	Reserved, must be cleared.
19–16 PETB	Prime endpoint transmit buffer. For each endpoint, a corresponding bit requests that a buffer be prepared for a transmit operation to respond to a USB IN/INTERRUPT transaction. Software must write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware automatically uses this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware clears this bit when associated endpoint(s) is (are) successfully primed. **Note:** These bits are momentarily set by hardware during hardware re-priming operations when a dTD retires, and the dQH updates.
15–4	Reserved, must be cleared.
3–0 PERB	Prime endpoint receive buffer. For each endpoint, a corresponding bit requests that a buffer be prepared for a receive operation to respond to a USB OUT transaction. Software must write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware automatically uses this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware clears this bit when associated endpoint(s) is (are) successfully primed. **Note:** These bits are momentarily set by hardware during hardware re-priming operations when a dTD retires, and the dQH updates.

10.3.4.19 Endpoint Flush Register (EPFLUSH)

This register is not defined in the EHCI specification. This register used only in device mode.

Address: 0xFC0B_01B4 (EPFLUSH)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	FETB				0	0	0	0	0	0	0	0	0	0	0	0	FERB			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-35. Endpoint Flush Register (EPFLUSH)

Table 10-38. EPFLUSH Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–16 FETB	Flush endpoint transmit buffer. Writing a 1 to a bit in this field causes the associated endpoint to clear any primed buffers. If a packet is in progress for an associated endpoint, that transfer continues until completion. Hardware clears this register after the endpoint flush operation is successful.
15–4	Reserved, must be cleared.
3–0 FERB	Flush endpoint receive buffer. Writing a 1 to a bit in this field causes the associated endpoint to clear any primed buffers. If a packet is in progress for an associated endpoint, that transfer continues until completion. Hardware clears this register after the endpoint flush operation is successful. FERB[3] corresponds to endpoint 3.

10.3.4.20 Endpoint Status Register (EPSR)

This register is not defined in the EHCI specification. This register is only used in device mode.

Address: 0xFC0B_01B8 (EPSR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	ETBR				0	0	0	0	0	0	0	0	0	0	0	0	ERBR			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-36. Endpoint Status Register (EPSR)

Table 10-39. EPSR Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–16 ETBR	Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. The hardware sets this bit in response to receiving a command from a corresponding bit in the EPPRIME register. A constant delay exists between setting a bit in the EPPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the EPPRIME register. USB reset, USB DMA system, or EPFLUSH register clears the buffer ready. ETBR[3] (bit 19) corresponds to endpoint 3. **Note:** Hardware momentarily clears these bits during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.
15–4	Reserved, must be cleared.
3–0 ERBR	Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. The hardware sets this bit in response to receiving a command from a corresponding bit in the EPPRIME register. A constant delay exists between setting a bit in the EPPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the EPPRIME register. USB reset, USB DMA system, or EPFLUSH register clears the buffer ready. ERBR[3] (bit 19) corresponds to endpoint 3. **Note:** Hardware momentarily clears these bits during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

10.3.4.21 Endpoint Complete Register (EPCOMPLETE)

This register is not defined in the EHCI specification. This register is used only in device mode.

Address: 0xFC0B_01BC (EPCOMPLETE) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	ETCE				0	0	0	0	0	0	0	0	0	0	0	0	ERCE			
W													w1c																w1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-37. Endpoint Complete Register (EPCOMPLETE)

Table 10-40. EPCOMPLETE Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–16 ETCE	Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurs and software must read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the transfer descriptor, this bit is set simultaneously with the USBINT. Writing a 1 clears the corresponding bit in this register. ETCE[3] (bit 19) corresponds to endpoint 3.

Table 10-40. EPCOMPLETE Field Descriptions (continued)

Field	Description
15–4	Reserved, must be cleared
3–0 ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurs and software must read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the transfer descriptor, this bit is set simultaneously with the USBINT. Writing a 1 clears the corresponding bit in this register. ERCE[3] corresponds to endpoint 3.

10.3.4.22 Endpoint Control Register 0 (EPCR0)

This register is not defined in the EHCI specification. Every device implements endpoint 0 as a control endpoint.

Address: 0xFC0B_01C0 (EPCR0)

Access: User 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
R	0	0	0	0	0	0	0	0	TXE	0	0	0	TXT		0	TXS	0	0	0	0	0	0	0	0	RXE	0	0	0	RXT		0	RXS
W																																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 10-38. Endpoint Control 0 (EPCR0)

Table 10-41. EPCR0 Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23 TXE	TX endpoint enable. Endpoint zero is always enabled. 1 Enable
22–20	Reserved, must be cleared.
19–18 TXT	TX endpoint type. Endpoint zero is always a control endpoint. 00 Control
17	Reserved, must be cleared.
16 TXS	TX endpoint stall. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears the bit or it automatically clears upon receipt of a new SETUP request. 0 Endpoint OK 1 Endpoint stalled
15–8	Reserved, must be cleared.
7 RXE	RX endpoint enable. Endpoint zero is always enabled. 1 Enabled.
6–4	Reserved, must be cleared.
3–2 RXT	RX endpoint type. Endpoint zero is always a control endpoint. 00 Control

Table 10-41. EPCR0 Field Descriptions (continued)

Field	Description
1	Reserved, must be cleared.
0 RXS	RX endpoint stall. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears the bit or it automatically clears upon receipt of a new SETUP request. 0 Endpoint OK 1 Endpoint stalled

10.3.4.23 Endpoint Control Register *n* (EPCR*n*)

These registers are not defined in the EHCI specification. There is an EPCR*n* register for each endpoint in a device.

Address: 0xFC0B_01C4 (EPCR1)
0xFC0B_01C8 (EPCR2)
0xFC0B_01CA (EPCR3)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	TXE	0	TXI	0	TXT		TXD	TXS
W										TXR						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RXE	0	RXI	0	RXT		RXD	RXS
W										RXR						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-39. Endpoint Control Registers (EPCR*n*)

Table 10-42. EPCR*n* Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23 TXE	TX endpoint enable. 0 Disabled 1 Enabled
22 TXR	TX data toggle reset. When a configuration event is received for this Endpoint, software must write a 1 to this bit to synchronize the data PID's between the host and device. This bit is self-clearing.
21 TXI	TX data toggle inhibit. This bit is used only for test and should always be written as 0. Writing a 1 to this bit causes this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet. 0 PID sequencing enabled. 1 PID sequencing disabled.
20	Reserved, must be cleared.

Table 10-42. EPCR*n* Field Descriptions (continued)

Field	Description
19–18 TXT	TX endpoint type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt **Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
17 TXD	TX endpoint data source. This bit should always be written as 0, which selects the dual port memory/DMA engine as the source.
16 TXS	TX endpoint stall. This bit sets automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It clears automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears this bit clears or automatically clears as above. 0 Endpoint OK 1 Endpoint stalled
15–8	Reserved, must be cleared.
7 RXE	RX endpoint enable. 0 Disabled 1 Enabled
6 RXR	RX data toggle reset. When a configuration event is received for this endpoint, software must write a 1 to this bit to synchronize the data PIDs between the host and device. This bit is self-clearing.
5 RXI	RX data toggle inhibit. This bit is only for testing and should always be written as 0. Writing a 1 to this bit causes this endpoint to ignore the data toggle sequence and always accept data packets regardless of their data PID. 0 PID sequencing enabled 1 PID sequencing disabled
4	Reserved, must be cleared.
3–2 RXT	RX endpoint type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt **Note:** When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
1 RXD	RX endpoint data sink. This bit should always be written as 0, which selects the dual port memory/DMA engine as the sink.
0 RXS	RX endpoint stall. This bit sets automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It clears automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint, Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears this bit or automatically clears as above, 0 Endpoint OK 1 Endpoint stalled

10.4 Functional Description

This module can be broken down into functional sub-blocks as described below.

10.4.1 System Interface

The system interface block contains all the control and status registers to allow a core to interface to the module. These registers allow the processor to control the configuration and ascertain the capabilities of the module and, they control the module’s operation.

10.4.2 DMA Engine

The USB module contains a local DMA engine. It is responsible for moving all of the data transferred over the USB between the module and system memory. Like the system interface block, the DMA engine block uses a simple synchronous bus signaling protocol.

The DMA controllers must access control information and packet data from system memory. Control information is contained in link list based queue structures. The DMA controllers have state machines able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers performed by the host controller, including the split-transaction requests that allow an EHCI controller to direct packets to FS and LS speed devices. In device mode, data structures are similar to those in the EHCI specification and used to allow device responses to be queued for each of the active pipes in the device.

10.4.3 FIFO RAM Controller

The FIFO RAM controller is used for context information and to control FIFOs between the protocol engine and the DMA controller. These FIFOs decouple the system processor/memory bus requests from the extremely tight timing required by USB.

The use of the FIFO buffers differs between host and device mode operation. In host mode, a single data channel maintains in each direction through the buffer memory. In device mode, multiple FIFO channels maintain for each of the active endpoints in the system.

In host mode, the USB OTG modules use 16-byte transmit buffers and 16-byte receive buffers. For the USB OTG module, device operation uses a single 16-byte receive buffer and a 16-byte transmit buffer for each endpoint.

10.4.4 Physical Layer (PHY) Interface

Readers should familiarize themselves with chapter 7 of the *Universal Serial Bus Specification, Revision 2.0*. The USB OTG modules contain an on-chip digital to analog transceiver (XCVR) for DP and DN USB network communication. The USB module defaults to FS XCVR operation and can communicate in LS. The USB OTG module may interface to any ULPI compatible PHY as well.

Due to pin-count limitations the USB module only supports certain combinations of PHY interfaces and USB functionality. Refer to the Table 10-43 for more information.

Table 10-43. USB Network Speed and Required Physical Interface

USB Mode and Speed	DP and DN On-Chip Analog XCVR Active	I^2C	FEC	External Integrated Circuit Required
USB Host FS/LS	Yes	No	Yes	See Section 10.4.4.1, “USB On-Chip Transceiver Required External Components”
USB Device FS	Yes	No	Yes	See Section 10.4.4.1, “USB On-Chip Transceiver Required External Components”
Host/Device ULPI HS/FS	No	Yes	No	Maxim

10.4.4.1 USB On-Chip Transceiver Required External Components

USB system operation does not require external components. However, the recommended method ensures driver output impedance, eye diagram, and V_{BUS} cable fault tolerance requirements are met. The recommended method is for the DM and DP I/O pads to connect through series resistors (approximately 33 Ω each) to the USB connector on the application printed circuit board (PCB). Additionally, signal quality optimizes when these 33 Ω resistors are mounted close to the processor rather than closer to the USB board level connector.

NOTE

Internal pull-down resistors are included that keep the DP and DM ports in a known quiescent state when the USB port is not used or when a USB cable is not connected.

Also included is an internal 1.5k Ω pull-up resistor on DP controlled by the CCM. (See Chapter 11, “Chip Configuration Module (CCM),” for more details.) This allows the OTG module to operate in full-speed device operation. Host operation requires this internal resistor to be disabled via the CCM, and 15k Ω external resistors to connect from DP and DM signals to ground.

10.5 Initialization/Application Information

10.5.1 Host Operation

Enhanced Host Controller Interface (EHCI) Specification defines the general operational model for the USB module in host mode. The EHCI specification describes the register-level interface for a host controller for USB Revision 2.0. It includes a description of the hardware/software interface between system software and host controller hardware. The next section has information about the initialization of the USB modules; however, full details of the EHCI specification are beyond the scope of this document.

10.5.1.1 Host Controller Initialization

After initial power-on or module reset (via the USBCMD[RST] bit), all of the operational registers are at default values, as illustrated in the register memory map in Table 10-4.

To initialize the host controller, software must:

1. Optionally set streaming disable in the USBMODE[SDIS] bit.
2. Optionally modify the BURSTSIZE register.
3. Program the PORTSC1[PTS] field if using a non-ULPI PHY.
4. Optionally write the appropriate value to the USBINTR register to enable the desired interrupts.
5. Set the USBMODE[CM] field to enable host mode, and set the USBMODE[ES] bit for big endian operation.
6. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable), and turn the controller on by setting the USBCMD[RS] bit.
7. Enable external VBUS supply. The exact steps required for initialization depend on the external hardware used to supply the 5V VBUS power.
8. Set the PORTSC[PP] bit.

At this point, the host controller is up and running and the port registers begin reporting device connects. System software can enumerate a port through the reset process (port is in the enabled state).

To communicate with devices via the asynchronous schedule, system software must write the ASYNCLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by setting the asynchronous schedule enable (ASE) bit in the USBCMD register. To communicate with devices via the periodic schedule, system software must enable the periodic schedule by setting the periodic schedule enable (PSE) bit in the USBCMD register. Schedules can be turned on before the first port is reset and enabled.

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

10.5.2 Device Data Structures

This section defines the interface data structures used to communicate control, status, and data between device controller driver (DCD) software and the device controller.The interface consists of device queue heads and transfer descriptors.

NOTE

Software must ensure that data structures do not span a 4K-page boundary.

The USB OTG uses an array of device endpoint queue heads to organize device transfers. As shown in Figure 10-40, there are two endpoint queue heads in the array for each device endpoint—one for IN and one for OUT. The EPLISTADDR provides a pointer to the first entry in the array.

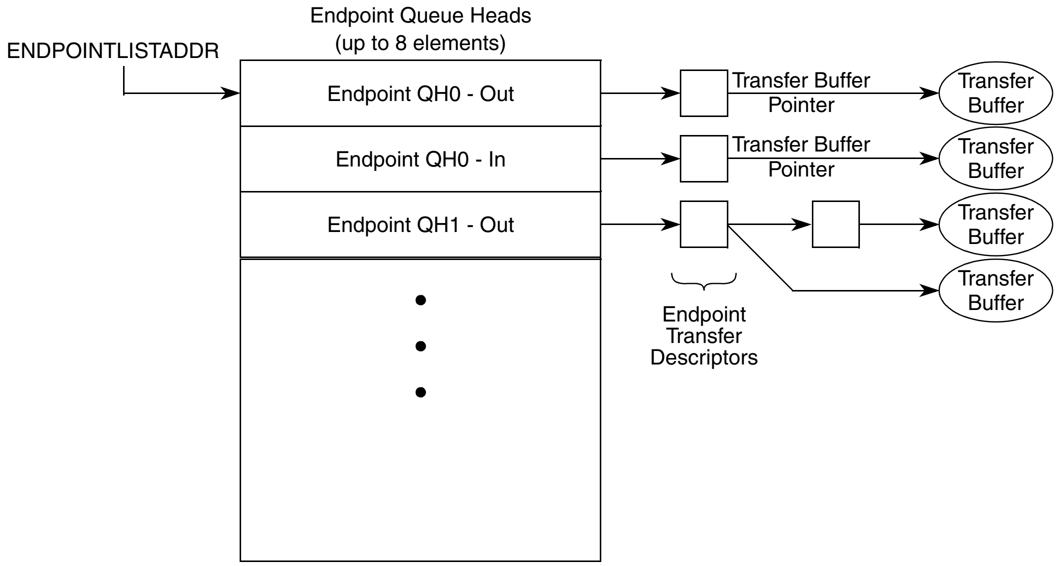


Figure 10-40. End Point Queue Head Organization

10.5.2.1 Endpoint Queue Head

All transfers are managed in the device endpoint queue head (dQH). The dQH is a 48-byte data structure, but must align on 64-byte boundaries. During priming of an endpoint, the dTD (device transfer descriptor) copies into the overlay area of the dQH, which starts at the nextTD pointer longword and continues through the end of the buffer pointers longwords. After a transfer is complete, the dTD status longword updates in the dTD pointed to by the currentTD pointer. While a packet is in progress, the overlay area of the dQH acts as a staging area for the dTD so the device controller can access needed information with minimal latency.

Figure 10-41 shows the endpoint queue head structure.

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	offset
Mult		ZLT	0	0	Maximum Packet Length											IOS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00
Current dTD Pointer																											0	0	0	0	0	0x04
Next dTD Pointer																											0	0	0	0	T	0x08[1]
0	0	Total Bytes														IOC	0	0	0	MultO		0	0	Status								0x0C[1]
Buffer Pointer (Page 0)																				Current Offset												0x10[1]
Buffer Pointer (Page 1)																				Reserved												0x14[1]
Buffer Pointer (Page 2)																				Reserved												0x18[1]
Buffer Pointer (Page 3)																				Reserved												0x1C[1]
Buffer Pointer (Page 4)																				Reserved												0x20[1]
Reserved																																0x24
Setup Buffer Bytes 3–0																																0x28
Setup Buffer Bytes 7–4																																0x2C

▨ Device controller read/write; all others read-only.

Figure 10-41. Endpoint Queue Head Layout

[1] Offsets 0x08 through 0x20 contain the transfer overlay.

10.5.2.1.1 Endpoint Capabilities/Characteristics (Offset = 0x0)

This longword specifies static information about the endpoint. In other words, this information does not change over the lifetime of the endpoint. DCD software must not attempt to modify this information while the corresponding endpoint is enabled.

Table 10-44. Endpoint Capabilities/Characteristics

Field	Description
31–30 Mult	Mult. This field indicates the number of packets executed per transaction description as given by: 00 Execute N Transactions as demonstrated by the USB variable length packet protocol where N computes using the Maximum Packet Length (dQH) and the Total Bytes field (dTD) 01 Execute 1 Transaction. 10 Execute 2 Transactions. 11 Execute 3 Transactions. **Note:** Non-ISO endpoints must set Mult equal to 00. ISO endpoints must set Mult equal to 01, 10, or 11 as needed.

Table 10-44. Endpoint Capabilities/Characteristics (continued)

Field	Description
29 ZLT	Zero length termination select. This bit is ignored in isochronous transfers. Clearing this bit enables the hardware to automatically append a zero length packet when the following conditions are true: • The packet transmitted equals maximum packet length • The dTD has exhausted the field Total Bytes After this the dTD retires. When the device is receiving, if the last packet length received equals the maximum packet length and the total bytes is zero, it waits for a zero length packet from the host to retire the current dTD. Setting this bit disables the zero length packet. When the device is transmitting, the hardware does not append any zero length packet. When receiving, it does not require a zero length packet to retire a dTD whose last packet was equal to the maximum packet length packet. The dTD is retired as soon as Total Bytes field goes to zero, or a short packet is received. 0 Enable zero length packet (default). 1 Disable the zero length packet. **Note:** Each transfer is defined by one dTD, so the zero length termination is for each dTD. In some software application cases, the logic transfer does not fit into only one dTD, so it does not make sense to add a zero length termination packet each time a dTD is consumed. On those cases we recommend to disable the ZLT feature, and use software to generate the zero length termination.
28–27	Reserved. Reserved for future use and must be cleared.
26–16 Maximum Packet Length	Maximum packet length. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15 IOS	Interrupt on setup (IOS). This bit used on control type endpoints indicates if USBSTS[UI] is set in response to a setup being received.
14–0	Reserved. Reserved for future use and must be cleared.

10.5.2.1.2 Current dTD Pointer (Offset = 0x4)

The device controller uses the current dTD pointer to locate transfer in progress. This word is for USB OTG (hardware) use only and should not be modified by DCD software.

Table 10-45. Current dTD Pointer

Field	Description
31–5 Current dtd	Current dtd. This field is a pointer to the dTD represented in the transfer overlay area. This field is modified by the device controller to next dTD pointer during endpoint priming or queue advance.
4–0	Reserved. Reserved for future use and must be cleared.

10.5.2.1.3 Transfer Overlay (Offset = 0x8–0x20)

The seven longwords in the overlay area represent a transaction working space for the device controller. The general operational model is that the device controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, it does not read the associated endpoint.

After an endpoint is readied, the dTD is copied into this queue head overlay area by the device controller. Until a transfer expires, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller writes the results back to the original transfer descriptor and advance the queue.

See Section 10.5.2.2, “Endpoint Transfer Descriptor (dTD),” for a description of the overlay fields.

10.5.2.1.4 Setup Buffer (Offset = 0x28–0x2C)

The set-up buffer is dedicated storage for the 8-byte data that follows a set-up PID. Refer to Section 10.5.3.4.4, “Control Endpoint Operation” for information on the procedure for reading the setup buffer

NOTE

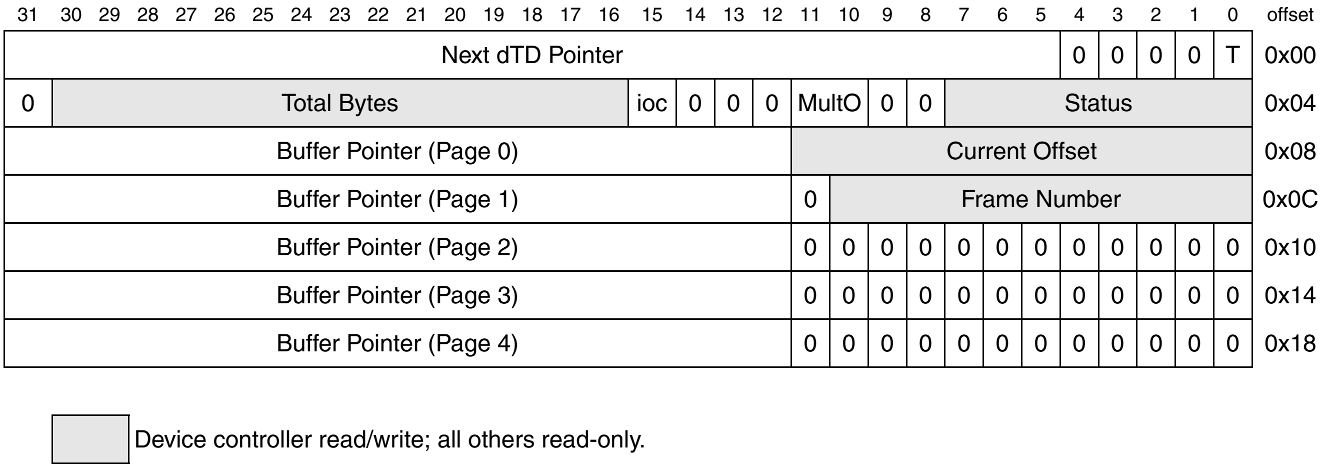
Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head receives setup data packets.

Table 10-46. Multiple Mode Control

longword	Field	Description
1	31–0 Setup Buffer 0	Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and is written by the device controller software reads.
2	31–0 Setup Buffer 1	Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and is written by the device controller software reads.

10.5.2.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data sent/received for a given transfer. The DCD software should not attempt to modify any field in an active dTD except the next dTD pointer, which must be modified only as described in Section 10.5.3.6, “Managing Transfers with Transfer Descriptors.”



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	offset
Next dTD Pointer																											0	0	0	0	T	0x00
0	Total Bytes															ioc	0	0	0	MultO		0	0	Status								0x04
Buffer Pointer (Page 0)																				Current Offset												0x08
Buffer Pointer (Page 1)																				0	Frame Number											0x0C
Buffer Pointer (Page 2)																				0	0	0	0	0	0	0	0	0	0	0	0	0x10
Buffer Pointer (Page 3)																				0	0	0	0	0	0	0	0	0	0	0	0	0x14
Buffer Pointer (Page 4)																				0	0	0	0	0	0	0	0	0	0	0	0	0x18

▨ Device controller read/write; all others read-only.

Figure 10-42. Endpoint Transfer Descriptor (dTD)

10.5.2.2.1 Next dTD Pointer (Offset = 0x0)

The next dTD pointer is used to point the device controller to the next dTD in the linked list.

Table 10-47. Next dTD Pointer

Field	Description
31–5 Next dTD pointer	Next dTD pointer. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals [31:5], respectively.
4–1	Reserved. Reserved for future use and must be cleared.
0 T	Terminate. This bit indicates to the device controller no more valid entries exist in the queue. 0=Pointer is valid (points to a valid transfer element descriptor). 1=pointer is invalid.

10.5.2.2.2 dTD Token (Offset = 0x4)

The dTD token is used to specify attributes for the transfer including the number of bytes to read or write and the status of the transaction.

Table 10-48. dTD Token

Field	Description
31	Reserved. Reserved for future use and must be cleared.
30–16 Total Bytes	Total bytes. This field specifies the total number of bytes moved with this transfer descriptor. This field decrements by the number of bytes actually moved during the transaction and only on the successful completion of the transaction. The maximum value software may store in the field is 5*4K(0x5000). This is the maximum number of bytes 5 page pointers can access. Although possible to create a transfer up to 20K, this assumes the first offset into the first page is 0. When the offset cannot be predetermined, crossing past the fifth page can be guaranteed by limiting the total bytes to 16K**. Therefore, the maximum recommended transfer is 16K (0x4000). **Note:** Larger transfer sizes can be supported, but require disabling ZLT and using multiple dTDs. If the value of the field is 0 when the host controller fetches this transfer descriptor (and the active bit is set), the device controller executes a zero-length transaction and retires the transfer descriptor. For IN transfers it is not a requirement for total bytes to transfer be an even multiple of the maximum packet length. If software builds such a transfer descriptor for an IN transfer, the last transaction is always less than maximum packet length. For OUT transfers the total bytes must be evenly divisible by the maximum packet length.
15 IOC	Interrupt on complete. Indicates if USBSTS[UI] is set in response to device controller finished with this dTD.
14–12	Reserved. Reserved for future use and must be cleared.

Table 10-48. dTD Token (continued)

<table>
<tr><th>Field</th><th colspan="2">Description</th></tr>
<tr><td>11–10
MultO</td><td colspan="2">Multiplier Override. This field can possibly transmit-ISOs (ISO-IN) to override the multiplier in the QH. This field must be 0 for all packet types not transmit-ISO.
For example, if QH.MULT equals 3; Maximum packet size equals 8; Total Bytes equals 15; MultiO equals 0 [default], then three packets are sent: {Data2(8); Data1(7); Data0(0)}.
If QH.MULT equals 3; Maximum packet size equals 8; Total Bytes equals 15; MultO equals 2, then two packets are sent: {Data1(8); Data0(7)}
For maximal efficiency, software must compute MultO equals greatest integer of (Total Bytes / Max. Packet Size) except for the case when Total Bytes equals 0; then MultO must be 1.
Note: Non-ISO and Non-TX endpoints must set MultO equals 00.</td></tr>
<tr><td>9–8</td><td colspan="2">Reserved. Reserved for future use and must be cleared.</td></tr>
<tr><td rowspan="8">7–0
Status</td><td colspan="2">Status. Device controller communicates individual command execution states back to the DCD software. This field contains the status of the last transaction performed on this dTD. The bit encodings are:</td></tr>
<tr><td>**Bit**</td><td>**Status Field Description**</td></tr>
<tr><td>7</td><td>Active. Set by software to enable the execution of transactions by the device controller.</td></tr>
<tr><td>6</td><td>Halted. Set by the device controller during status updates to indicate a serious error has occurred at the device/endpoint addressed by this dTD. Any time a transaction results in the halted bit being set, the active bit is also cleared.</td></tr>
<tr><td>5</td><td>Data Buffer Error. Set by the device controller during status update to indicate the device controller is unable to maintain the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run).</td></tr>
<tr><td>4</td><td>Reserved.</td></tr>
<tr><td>3</td><td>Transaction Error. Set by the device controller during status update in case the device did not receive a valid response from the host (time-out, CRC, bad PID).</td></tr>
<tr><td>2–0</td><td>Reserved.</td></tr>
</table>

10.5.2.2.3 dTD Buffer Page Pointer List (Offset = 0x8–0x18)

The last five longwords of a device element transfer descriptor are an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

Table 10-49. Buffer Page Pointer List

Field	Description
31–12 Buffer Pointer	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems typically set the buffer pointers to a series of incrementing integers.
0;11–0 Current Offset	Current Offset. Offset into the 4kB buffer where the packet begins.
1;10–0 Frame Number	Frame Number. Written by the device controller to indicate the frame number a packet finishes in. Typically correlates relative completion times of packets on an ISO endpoint.

10.5.3 Device Operation

The device controller performs data transfers using a set of linked list transfer descriptors pointed to by a queue head. The next sections explain the use of the device controller from the device controller driver (DCD) point-of-view and further describe how specific USB bus events relate to status changes in the device controller programmer's interface.

10.5.3.1 Device Controller Initialization

After hardware reset, USB OTG is disabled until the run/stop bit in the USBCMD register is set. At minimum, it is necessary to have the queue heads set up for endpoint 0 before the device attach occurs. Shortly after the device is enabled, a USB reset occurs followed by setup packet arriving at endpoint 0. A queue head must be prepared so the device controller can store the incoming setup packet.

To initialize a device, the software must:

1. Optionally set streaming disable in the USBMODE[SDIS] bit.
2. Optionally modify the BURSTSIZE register.
3. Program the PORTSC1[PTS] field if using a non-ULPI PHY.
4. Write the appropriate value to the USBINTR to enable the desired interrupts. For device operation, setting UE, UEE, PCE, URE, and SLE is recommended.

   For a list of available interrupts, refer to Section 10.3.4.3, “USB Interrupt Enable Register (USBINTR),” and Section 10.3.4.2, “USB Status Register (USBSTS).”
5. Set the USBMODE[CM] field to enable device mode, and set the USBMODE[ES] bit for big endian operation.
6. Optionally write the USBCMD register to set the desired interrupt threshold.
7. Set USBMODE[SLOM] to disable setup lockouts.
8. Initialize the EPLISTADDR.
9. Create two dQHs for endpoint 0—one for IN transactions and one for OUT transactions.

   For information on device queue heads, refer to Section 10.5.2.1, “Endpoint Queue Head.”
10. Set the CCM’s UOCSR[BVLD] bit to allow device to connect to a host.
11. Set the USBCMD[RS] bit.

After the run/stop bit is set, a device reset occurs. The DCD must monitor the reset event and set the DEVICEADDR and EPCR*n* registers, and adjust the software state as described in Section 10.5.3.2.1, “Bus Reset.”

NOTE

> Endpoint 0 is a control endpoint only and does not need to configured using the EPCR0 register.

It is not necessary to initially prime endpoint 0 because the first packet received is always a setup packet. The contents of the first setup packet requires a response in accordance with USB device framework command set.

10.5.3.2 Port State and Control

From a chip or system reset, the USB OTG module enters the powered state. A transition from the powered state to the attach state occurs when the USBCMD[RS] bit is set. After receiving a reset on the bus, the port enters the defaultFS or defaultHS state in accordance with the protocol reset described in Appendix C.2 of the *Universal Serial Bus Specification, Revision 2.0*. Figure 10-43 depicts the state of a USB 2.0 device.

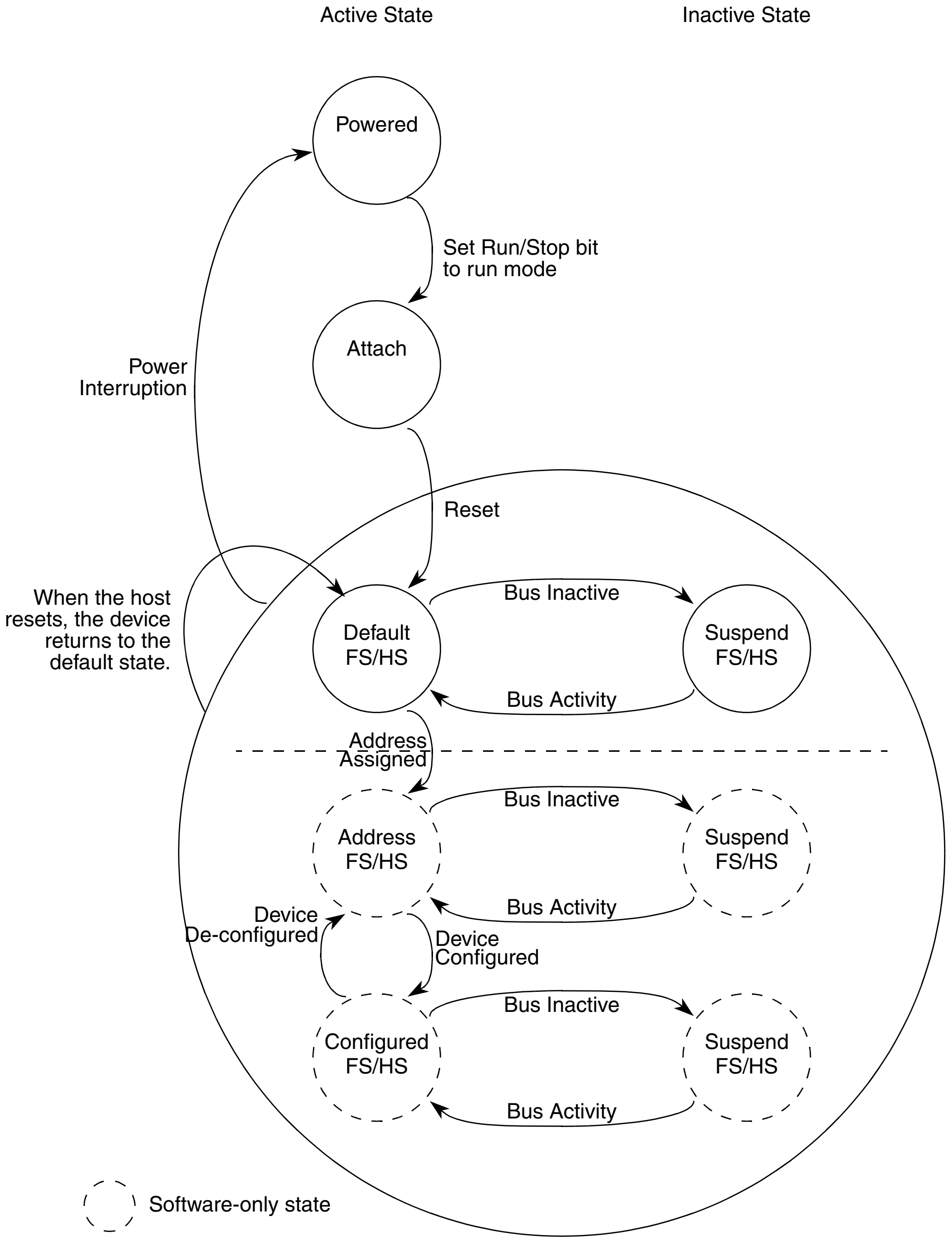


Figure 10-43. USB 2.0 Device States

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB OTG, and they are communicated to the DCD using these status bits:

Table 10-50. Device Controller State Information Bits

Bit	Register
DC Suspend ()	USBSTS
USB Reset Received (URI)	USBSTS
Port Change Detect (PCI)	USBSTS
High-Speed Port (PSPD)	PORTSC*n*

DCD software must maintain a state variable to differentiate between the defaultFS/HS state and the address/configured states. Change of state from default to the address and configured states is part of the enumeration process described in the device framework section of the USB 2.0 specification.

As a result of entering the address state, the DCD must program the device address register (DEVICEADDR).

Entry into the configured state indicates that all endpoints to be used in the operation of the device have been properly initialized by programming the EPCR*n* registers and initializing the associated queue heads.

10.5.3.2.1 Bus Reset

The host uses a bus reset to initialize downstream devices. When a bus reset is detected, USB OTG controller renegotiates its attachment speed, resets the device address to 0, and notifies the DCD by interrupt (assuming the USB reset interrupt enable is set). After a reset is received, all endpoints (except endpoint 0) are disabled and the device controller cancels any primed transactions. The concept of priming is clarified below, but when a reset is received, the DCD must perform:

1. Clear all setup token semaphores by reading the EPSETUPSR register and writing the same value back to the EPSETUPSR register.
2. Clear all the endpoint complete status bits by reading the EPCOMPLETE register and writing the same value back to the EPCOMPLETE register.
3. Cancel all primed status by waiting until all bits in the EPPRIME are 0 and then writing 0xFFFF_FFFF to EPFLUSH.
4. Read the reset bit in the PORTSC*n* register and make sure it remains active. A USB reset occurs for a minimum of 3 ms and the DCD must reach this point in the reset clean-up before end of the reset occurs, otherwise a hardware reset of the device controller is recommended (rare).
   a) Setting USBCMD[RST] bit can perform a hardware reset.

NOTE

A hardware reset causes the device to detach from the bus by clearing the USBCMD[RS] bit. Therefore, the DCD must completely re-initialize the USB OTG after a hardware reset.

5. Free all allocated dTDs because the device controller no longer executes them. If this is the first time the DCD processes a USB reset event, it is likely w3a4no dTDs have been allocated.
6. At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a port change detect is indicated.

7. After a port change detect, the device has reached the default state and the DCD can read the PORTSC*n* register to determine if the device operates in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the chapter 9 Device Framework of the USB specification.

In some applications, it may not be possible to enable one or more pipes while in FS mode. Beyond the data rate issue, there is no difference in DCD operation between FS and HS modes.

10.5.3.2.2 Suspend/Resume

To conserve power, USB OTG module automatically enters the suspended state when no bus traffic is observed for a specified period. When suspended, the module maintains any internal status, including its address and configuration. Attached devices must be prepared to suspend any time they are powered, regardless if they are assigned a non-default address, are configured, or neither. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled.

The USB OTG module exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB OTG is capable of remote wake-up signaling. When the USB OTG is reset, remote wake-up signaling must be disabled.

Suspend Operational Model

The USB OTG moves into the suspend state when suspend signaling is detected or activity is missing on the upstream port for more than a specific period. After the device controller enters the suspend state, an interrupt notifies the DCD (assuming device controller suspend interrupt is enabled, USBINTR[SLE] is set). When the PORTSC*n*[SUSP] is set, the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low power operation. Find information on the bus power limits in suspend state in USB 2.0 specification.

Resume

If the USB OTG is suspended, its operation resumes when any non-idle signaling is received on its upstream facing port. In addition, the USB OTG can signal the system to resume operation by forcing resume signaling to the upstream port. Setting the PORTSC*n*[FPR] bit while the device is in suspend state sends resume signaling upstream. Sending resume signal to an upstream port should cause the host to issue resume signaling and bring the suspended bus segment (one more devices) back to the active condition.

NOTE

> Before use of resume signaling, the host must enable it by using the set feature command defined in chapter 9 Device Framework of the USB 2.0 specification.

10.5.3.3 Managing Endpoints

The USB 2.0 specification defines an endpoint (also called a device endpoint or an address endpoint) as a uniquely addressable portion of a USB device that can source or sink data in a communications channel between the host and the device. Combination of the endpoint number and the endpoint direction specifies endpoint address.

The channel between the host and an endpoint at a specific device represents a data pipe. Endpoint 0 for a device is always a control type data channel used for device discovery and enumeration. Other types of endpoints are supported by USB include bulk, interrupt, and isochronous. Each endpoint type has specific behavior related to packet response and error managing. Find more detail on endpoint operation in the USB 2.0 specification.

The USB OTG supports up to four endpoint specified numbers. The DCD can enable, disable, and configure each endpoint.

Each endpoint direction is essentially independent and can have differing behavior in each direction. For example, the DCD can configure endpoint 1-IN to be a bulk endpoint and endpoint 1-OUT to be an isochronous endpoint. This helps to conserve the total number of endpoints required for device operation. The only exception is that control endpoints must use both directions on a single endpoint number to function as a control endpoint. Endpoint 0, for example, is always a control endpoint and uses both directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum is four endpoint numbers (one for each endpoint direction used by the device controller), eight queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

10.5.3.3.1 Endpoint Initialization

After hardware reset, all endpoints except endpoint 0 are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to the appropriate EPCR*n* register. Each EPCR*n* is split into an upper and lower half. The lower half of EPCR*n* configures the receive or OUT endpoint, and the upper half configures the corresponding transmit or IN endpoint. Control endpoints must be configured the same in the upper and lower half of the EPCR*n* register; otherwise, behavior is undefined. Table 10-51 shows how to construct a configuration word for endpoint initialization.

Table 10-51. Device Controller Endpoint Initialization

Field	Value
Data Toggle Reset (TXR, RXR)	1 Synchronize the data PIDs
Data Toggle Inhibit (TXI, RXI)	0 PID sequencing disabled
Endpoint Type (TXT, RXT)	00 Control 01 Isochronous 10 Bulk 11 Interrupt
Endpoint Stall (TXS, RXS)	0 Not stalled

10.5.3.3.2 Stalling

There USB OTG has two occasions it may need to return to the host a STALL:

- The first is the functional stall, a condition set by the DCD as described in the USB 2.0 Device Framework chapter. A functional stall is used only on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the EPCR*n* register associated with the given endpoint and the given direction. In a functional stall condition, the device controller continues to return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.
- A protocol stall, unlike a function stall, is used on control endpoints and automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, DCD must enable the stall bits as a pair (TXS and RXS bits). A single write to the EPCR*n* register can ensure both stall bits are set at the same instant.

NOTE

Any write to the EPCR*n* register during operational mode must preserve the endpoint type field (perform a read-modify-write).

Table 10-52. Device Controller Stall Response Matrix

USB Packet	Endpoint Stall Bit	Effect on Stall bit	USB Response
SETUP packet received by a non-control endpoint.	N/A	None	STALL
IN/OUT/PING packet received by a non-control endpoint.	1	None	STALL
IN/OUT/PING packet received by a non-control endpoint.	0	None	ACK/NAK/NYET
SETUP packet received by a control endpoint.	N/A	Cleared	ACK
IN/OUT/PING packet received by a control endpoint	1	None	STALL
IN/OUT/PING packet received by a control endpoint.	0	None	ACK/NAK/NYET

10.5.3.3.3 Data Toggle

Data toggle maintains data coherency between host and device for any given data pipe. For more information on data toggle, refer to the USB 2.0 specification.

Data Toggle Reset

The DCD may reset the data toggle state bit and cause the data toggle sequence to reset in the device controller by setting the data toggle reset bit in the EPCR*n* register. This should only happen when configuring/initializing an endpoint or returning from a STALL condition.

Data Toggle Inhibit

This feature is for test purposes only and must never be used during normal device controller operation.

Setting the data toggle inhibit bit causes the USB OTG module to ignore the data toggle pattern normally sent and accepts all incoming data packets regardless of the data toggle state.

In normal operation, the USB OTG checks the DATA0/DATA1 bit against the data toggle to determine if the packet is valid. If the data PID does not match the data toggle state bit maintained by the device controller for that endpoint, the data toggle is considered not valid. If the data toggle is not valid, the device controller assumes the packet was already received and discards the packet (not reporting it to the DCD). To prevent the USB OTG from re-sending the same packet, the device controller responds to the error packet by acknowledging it with an ACK or NYET response.

10.5.3.4 Packet Transfers

The host initiates all transactions on the USB bus and in turn, the device must respond to any request from the host within the turnaround time stated in the USB 2.0 specification.

A USB host sends requests to the USB OTG in an order that can not be precisely predicted as a single pipeline, so it is not possible to prepare a single packet for the device controller to execute. However, the order of packet requests is predictable when the endpoint number and direction is considered. For example, if endpoint 3 (transmit direction) is configured as a bulk pipe, expect the host to send IN requests to that endpoint. This USB OTG module prepares packets for each endpoint/direction in anticipation of the host request. The process of preparing the device controller to send or receive data in response to host initiated transaction on the bus is referred to as priming the endpoint. This term appears throughout the documentation to describe the USB OTG operation so the DCD is built properly. Further, the term flushing describes the action of clearing a packet queued for execution.

10.5.3.4.1 Priming Transmit Endpoints

Priming a transmit endpoint causes the device controller to fetch the device transfer descriptor (dTD) for the transaction pointed to by the device queue head (dQH). After the dTD is fetched, it is stored in the dQH until the device controller completes the transfer described by the dTD. Storing the dTD in the dQH allows the device controller to fetch the operating context needed to manage a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTD, the leading data in the packet is stored in a FIFO in the device controller. This FIFO splits into virtual channels so the leading data can be stored for any endpoint up to the maximum number of endpoints configured at device synthesis time.

After a priming request is complete, an endpoint state of primed is indicated in the EPSR register. For a primed transmit endpoint, the device controller can respond to an IN request from the host and meet the stringent bus turnaround time of high-speed USB.

10.5.3.4.2 Priming Receive Endpoints

Priming receives endpoints identical to priming of transmit endpoints from the point of view of the DCD. The major difference in the operational model at the device controller is no data movement of the leading packet data because the data is to be received from the host.

As part of the architecture, the FIFO for the receive endpoints is not partitioned into multiple channels like the transmit FIFO. Thus, the size of the RX FIFO does not scale with the number of endpoints.

10.5.3.4.3 Interrupt/Bulk Endpoint Operation

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes handshake with a NAK unless the endpoint is primed. After the endpoint is primed, data delivery commences.

A dTD is retired by the device controller when the packets described in the transfer descriptor are completed. Each dTD describes N packets to transfer according to the USB variable length transfer protocol. The formula below and Table 10-53 describe how the device controller computes the number and length of the packets sent/received by the USB vary according to the total number of bytes and maximum packet length. See Section 10.5.2.1.1, “Endpoint Capabilities/Characteristics (Offset = 0x0),” for details on the ZLT bit.

With zero-length termination (ZLT) cleared:

N = INT(number of bytes/max. packet length) + 1

With zero-length termination (ZLT) set:

N = MAXINT(number of bytes/max. packet length)

Table 10-53. Variable Length Transfer Protocol Example (ZLT=0)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	—
512	256	3	256	256	0
512	512	2	512	0	—

Table 10-54. Variable Length Transfer Protocol Example (ZLT=1)

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	—
512	256	2	256	256	—
512	512	1	512	—	—

NOTE

The MULT field in the dQH must be set to 00 for bulk, interrupt, and control endpoints.

TX-dTD is complete when:

- All packets described in the dTD successfully transmit. Total bytes in dTD equal 0 when this occurs.

RX-dTD is complete when:

- All packets described in the dTD are successfully received. Total bytes in dTD equal 0 when this occurs.
- A short packet (number of bytes < maximum packet length) was received.
  This is a successful transfer completion; DCD must check the total bytes field in the dTD to determine the number of bytes remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) or (total bytes received > total bytes specified).
  This is an error condition. The device controller discards the remaining packet and set the buffer error bit in the dTD. In addition, the endpoint flushes and the USBERR interrupt becomes active.

NOTE

Disabling zero-length packet termination allows transfers larger than the total bytes field spanning across two or more dTDs.

Upon successful completion of the packet(s) described by the dTD, the active bit in the dTD is cleared and the next pointer is followed when the terminate bit is clear. When the terminate bit is set, USB OTG flushes the endpoint/direction and ceases operations for that endpoint/direction.

Upon unsuccessful completion of a packet (see long packet above), the dQH is left pointing to the dTD in error. To recover from this error condition, DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

NOTE

All packet level errors, such as a missing handshake or CRC error, are retried automatically by the device controller. There is no required interaction with the DCD for managing such errors.

Table 10-55. Interrupt/Bulk Endpoint Bus Response Matrix

Token Type	Stall	Not Primed	Primed	Underflow	Overflow
Setup	Ignore	Ignore	Ignore	N/A	N/A
In	STALL	NAK	Transmit	BS Error[1]	N/A
Out	STALL	NAK	Receive + NYET/ACK[2]	N/A	NAK
Ping	STALL	NAK	ACK	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

[1] Force bit stuff error

[2] NYET/ACK — NYET unless the transfer descriptor has packets remaining according to the USB variable length protocol then ACK.

10.5.3.4.4 Control Endpoint Operation

Setup Phase

All requests to a control endpoint begin with a setup phase followed by an optional data phase and a required status phase.

Setup packet managing:

- Disable setup lockout by setting the setup lockout mode bit (USBMODE[SLOM]), once at initialization. Setup lockout is not necessary when using the tripwire as described below.

NOTE

> Leaving the setup lockout mode cleared results in a potential compliance issue.

- After receiving an interrupt and inspecting EPSETUPSR to determine a setup packet was received on a particular pipe:

1. Write 1 to clear corresponding bit in EPSETUPSR.
2. Set the setup tripwire bit (USBCMD[SUTW]).
3. Duplicate contents of dQH.SetupBuffer into local software byte array.
4. Read the USBCMD[SUTW] bit. If set, continue; if cleared, goto 2)
5. Clear the USBCMD[SUTW] bit.
6. Poll until the EPSETUPSR bit clears.
7. Process setup packet using the local software byte array copy and execute status/handshake phases.

NOTE

> After receiving a new setup packet, status and/or handshake phases may remain pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

Data Phase

Following the setup phase, the DCD must create a device transfer descriptor for the data phase and prime the transfer.

After priming the packet, the DCD must verify a new setup packet is not received by reading the EPSETUPSR register immediately verifying that the prime had completed. A prime completes when the associated bit in the EPPRIME register is cleared and the associated bit in the EPSR register is set. If the EPPRIME bit goes to 0 and the EPSR bit is not set, the prime fails. This can only happen because of improper setup of the dQH, dTD, or a setup arriving during the prime operation. If a new setup packet is indicated after the EPPRIME bit is cleared, then the transfer descriptor can be freed and the DCD must re-interpret the setup packet.

Should a setup arrive after the data stage is primed, the device controller automatically clears the prime status (EPSR) to enforce data coherency with the setup packet.

NOTE

> Error managing of data phase packets is the same as bulk packets described previously.

Status Phase

Similar to the data phase, the DCD must create a transfer descriptor (with byte length equal zero) and prime the endpoint for the status phase. The DCD must also perform the same checks of the EPSETUPSR as described above in the data phase.

NOTE

> Error managing of status phase packets is the same as bulk packets described previously.

Control Endpoint Bus Response Matrix

Table 10-56 shows the device controller response to packets on a control endpoint according to the device controller state.

Table 10-56. Control Endpoint Bus Response Matrix

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Setup	ACK	ACK	ACK	N/A	SYSERR[1]	
In	STALL	NAK	Transmit	BS Error[2]	N/A	N/A
Out	STALL	NAK	Receive + NYET/ACK[3]	N/A	NAK	N/A
Ping	STALL	NAK	ACK	N/A	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore

[1] SYSERR — System error must never occur when the latency FIFOs are correctly sized and the DCD is responsive.

[2] Force bit stuff error

[3] NYET/ACK — NYET unless the transfer descriptor has packets remaining according to the USB variable length protocol then ACK.

10.5.3.4.5 Isochronous Endpoint Operation

Isochronous endpoints used for real-time scheduled delivery of data, and their operational model is significantly different than the host throttled bulk, interrupt, and control data pipes. Real time delivery by the USB OTG is accomplished by:

- Exactly MULT packets per (micro)frame are transmitted/received.

NOTE

> MULT is a two-bit field in the device queue head. Isochronous endpoints do not use the variable length packet protocol.

- NAK responses are not used. Instead, zero length packets are sent in response to an IN request to unprimed endpoints. For unprimed RX endpoints, the response to an OUT transaction is to ignore the packet within the device controller.
- Prime requests always schedule the transfer described in the dTD for the next (micro)frame. If ISO-dTD remains active after that frame, ISO-dTD holds ready until executed or canceled by the DCD.

The USB OTG in host mode uses the periodic frame list to schedule data exchanges to isochronous endpoints. The operational model for device mode does not use such a data structure. Instead, the same dTD used for control/bulk/interrupt endpoints is also used for isochronous endpoints. The difference is in the managing of the dTD.

The first difference between bulk and ISO-endpoints is that priming an ISO-endpoint is a delayed operation such that an endpoint becomes primed only after a SOF is received. After the DCD writes the prime bit, the prime bit clears as usual to indicate to software that the device controller completed a priming the dTD for transfer. Internal to the design, the device controller hardware masks that prime start until the next frame boundary. This behavior is hidden from the DCD, but occurs so the device controller can match the dTD to a specific (micro)frame.

Another difference with isochronous endpoints is that the transaction must wholly complete in a (micro)frame. After an ISO transaction is started in a (micro)frame, it retires the corresponding dTD when MULT transactions occur or the device controller finds a fulfillment condition.

The transaction error bit set in the status field indicates a fulfillment error condition. When a fulfillment error occurs, the frame after the transfer failed to complete wholly, and the device controller retires the current ISO-dTD and move to the next ISO-dTD.

Fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction stays primed indefinitely. This means it is up to software must discard transmit ISO-dTDs that pile up from a failure of the host to move the data.

Finally, the last difference with ISO packets is in the data level error managing. When a CRC error occurs on a received packet, the packet is not retried similar to bulk and control endpoints. Instead, the CRC is noted by setting the transaction error bit and the data is stored as usual for the application software to sort out.

- TX packet retired:
  - MULT counter reaches zero.
  - Fulfillment error (transaction error bit is set):
    - # packets occurred > 0 AND # packets occurred < MULT

NOTE

For TX-ISO, MULT counter can be loaded with a lesser value in the dTD multiplier override field. If the multiplier override field is zero, the MULT counter initializes to the multiplier in the QH.

- RX packet retired:
  - MULT counter reaches zero.
  - Non-MDATA data PID is received
  - Overflow error:
    - Packet received is > maximum packet length. (Buffer Error bit is set)
    - Packet received exceeds total bytes allocated in dTD. (Buffer Error bit is set)
  - Fulfillment error (Transaction Error bit is set):
    - # packets occurred > 0 AND # packets occurred < MULT
  - CRC error (Transaction Error bit is set)

NOTE

For ISO, when a dTD is retired, the next dTD is primed for the next frame. For continuous (micro)frame to (micro)frame operation, DCD must ensure the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

Isochronous Pipe Synchronization

When it is necessary to synchronize an isochronous data pipe to the host, the (micro)frame number (FRINDEX register) can act as a marker. To cause a packet transfer to occur at a specific (micro)frame number (N), the DCD must interrupt on SOF during frame N-1. When the FRINDEX equals N-1, the DCD must write the prime bit. The USB OTG primes the isochronous endpoint in (micro)frame N-1 so the device controller executes delivery during (micro)frame N.

CAUTION

Priming an endpoint towards the end of (micro)frame N-1 does not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N+1 if the device controller does not have enough time to complete the prime before the SOF for packet N is received.

Isochronous Endpoint Bus Response Matrix

Table 10-57. Isochronous Endpoint Bus Response Matrix

Token Type	Stall	Not Primed	Primed	Underflow	Overflow
Setup	STALL	STALL	STALL	N/A	N/A
In	NULL[1] Packet	NULL Packet	Transmit	BS Error[2]	N/A

Table 10-57. Isochronous Endpoint Bus Response Matrix (continued)

Token Type	Stall	Not Primed	Primed	Underflow	Overflow
Out	Ignore	Ignore	Receive	N/A	Drop Packet
Ping	Ignore	Ignore	Ignore	Ignore	Ignore
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore

[1] Zero length packet

[2] Force bit stuff error

10.5.3.5 Managing Queue Heads

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device transfer descriptor (dTD). An area of memory pointed to by EPLISTADDR contains a group of all dQH's in a sequential list (Figure 10-44). The even elements in the list of dQH's receive endpoints (OUT/SETUP) and the odd elements transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. After the dTD retires, it is no longer part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors because pointers no longer exist within the queue head after the dTD is retired (see Section 10.5.3.6.1, “Software Link Pointers”).

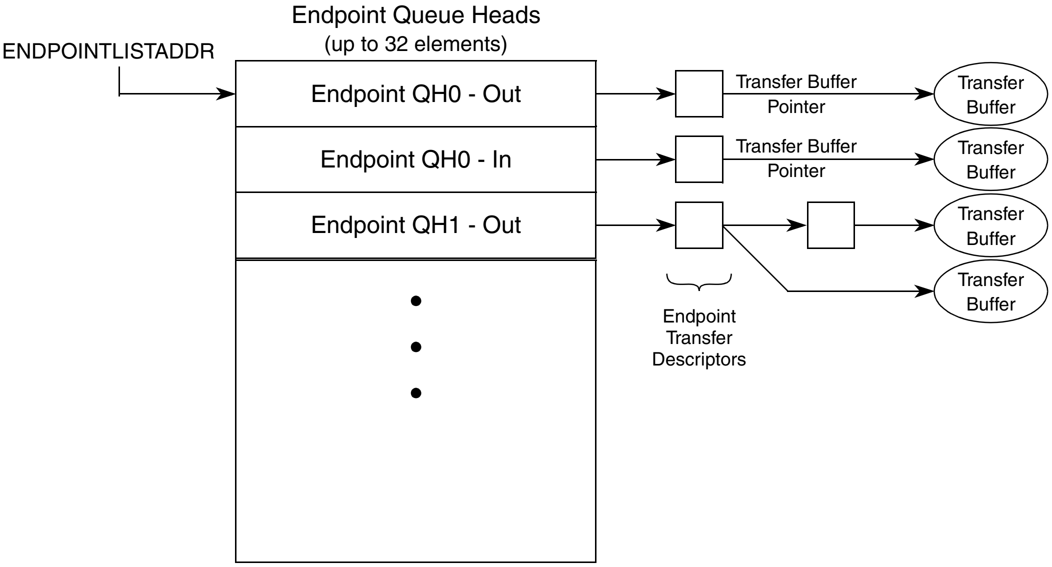


Figure 10-44. Endpoint Queue Head Diagram

In addition to current and next pointers and the dTD overlay examined in Section 10.5.3.4, “Packet Transfers,” the dQH also contains the following parameters for the associated endpoint: multipler, maximum packet length, and interrupt on setup. The next section includes demonstration of complete initialization of the dQH including these fields.

10.5.3.5.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the wMaxPacketSize field as required by the USB specification chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1,2, or 3 as required for bandwidth with the USB specification chapter 9 protocol. In FS mode, the multiplier field can only be 1 for ISO endpoints.
- Set the next dTD terminate bit field.
- Clear the active bit in the status field.
- Clear the halt bit in the status field.

NOTE

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTDs.

10.5.3.5.2 Setup Transfers Operation

As discussed in Section 10.5.3.4.4, “Control Endpoint Operation,” setup transfers require special treatment by the DCD. A setup transfer does not use a dTD, but instead stores the incoming data from a setup packet in an 8-byte buffer within the dQH.

Upon receiving notification of the setup packet, the DCD should manage the setup transfer by:

1. Copying setup buffer contents from dQH-RX to software buffer.
2. Acknowledging setup backup by writing a 1 to the corresponding bit in the EPSETUPSR register.

NOTE

The acknowledge must occur before continuing to process the setup packet. After acknowledge occurs, DCD must not attempt to access the setup buffer in dQH-RX. Only local software copy should be examined.

3. Checking for pending data or status dTD's from previous control transfers and flushing if any exist as discussed in Section 10.5.3.6.5, “Flushing/De-priming an Endpoint.”

NOTE

It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decoding setup packet and prepare data phase (optional) and status phase transfer as required by the USB specification chapter 9 or application specific protocol.

10.5.3.6 Managing Transfers with Transfer Descriptors

10.5.3.6.1 Software Link Pointers

It is necessary for the DCD software to maintain head and tail pointers for the linked list of dTDs for each respective queue head. This is necessary because the dQH only maintains pointers to the current working dTD and the next dTD executed. The operations described in the next section for managing dTDs assumes DCD can reference the head and tail of the dTD linked list.

NOTE

To conserve memory, the reserved fields at the end of the dQH can be used to store the head and tail pointers, but DCD must continue maintaining the pointers.

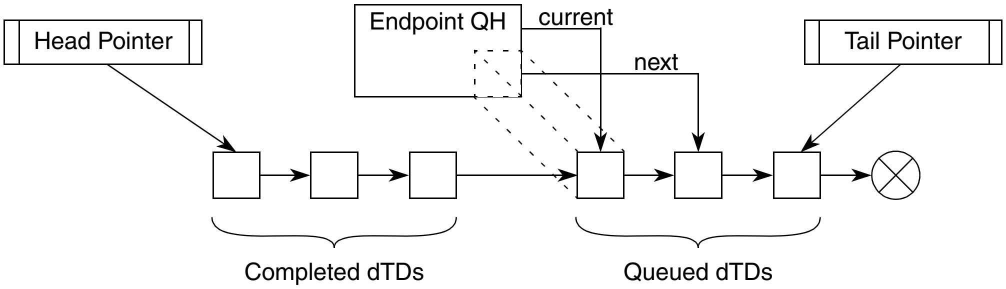


Figure 10-45. Software Link Pointers

NOTE

Check the status of each dTD to determine completed status.

10.5.3.6.2 Building a Transfer Descriptor

Before a transfer can be executed from the linked list, a dTD must be built to describe the transfer. Use the following procedure for building dTDs.

Allocate an 8-longword dTD block of memory aligned to 8-longword boundaries. The last 5 bits of the address must equal 00000.

Write the following fields:

1. Initialize the first 7 longwords to 0.
2. Set the terminate bit.
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete bit if desired.
5. Initialize the status field with the active bit set, and all remaining status bits cleared.
6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointers.

10.5.3.6.3 Executing a Transfer Descriptor

To safely add a dTD, the DCD must follow this procedure that manages the event where the device controller reaches the end of the dTD list. At the same time, a new dTD is added to the end of the list.

Determine whether the linked list is empty:

> Check the DCD driver to see if the pipe is empty (internal representation of the linked list should indicate if any packets are outstanding)

Case 1: Link list is empty

1. Write dQH next pointer AND dQH terminate bit to 0 as a single longword operation.
2. Clear active and halt bit in dQH (in case set from a previous error).
3. Prime endpoint by writing 1 to the correct bit position in the EPPRIME register.

Case 2: Link list is not empty

1. Add dTD to end of the linked list.
2. Read correct prime bit in EPPRIME - if set, DONE.
3. Set the USBCMD[ATDTW] bit.
4. Read correct status bit in EPSR, and store in a temporary variable for later.
5. Read the USBCMD[ATDTW] bit:

   If clear, go to 3.

   If set, continue to 6.
6. Clear the USBCMD[ATDTW] bit.
7. If status bit read in step 4 is 1 DONE.
8. If status bit read in step 4 is 0 then go to case 1, step 1.

10.5.3.6.4 Transfer Completion

After a dTD is initialized and the associated endpoint is primed, the device controller executes the transfer upon the host-initiated request. The DCD is notified with a USB interrupt if the interrupt-on-complete bit was set, or alternatively, the DCD can poll the endpoint complete register to determine when the dTD had been executed. After a dTD is executed, DCD can check the status bits to determine success or failure.

CAUTION

> Multiple dTDs can be completed in a single endpoint complete notification. After clearing the notification, the DCD must search the dTD linked list and retire all finished (active bit cleared) dTDs.

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0, Halted = 0, Transaction error = 0, Data buffer error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in Section 10.5.3.6.6, “Device Error Matrix.”

In addition to checking the status bit, the DCD must read the transfer bytes field to determine the actual bytes transferred. When a transfer is complete, the total bytes transferred decrements by the actual bytes transferred. For transmit packets, a packet is only complete after the actual bytes reaches zero. However, for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

10.5.3.6.5 Flushing/De-priming an Endpoint

It is necessary for the DCD to flush or de-prime endpoints during a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The DCD can use this procedure to stop a transfer in progress:

1. Set the corresponding bit(s) in the EPFLUSH register.
2. Wait until all bits in the EPFLUSH register are cleared.

NOTE

This operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.

3. Read the EPSR register to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now cleared. If the corresponding bits are set after step #2 has finished, flush failed as described below:

   In very rare cases, a packet is in progress to the particular endpoint when commanded to flush using EPFLUSH. A safeguard is in place to refuse the flush to ensure that the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush by repeating steps 1-3 until each endpoint successfully flushes.

10.5.3.6.6 Device Error Matrix

The following table summarizes packet errors not automatically managed by the USB OTG module.

Table 10-58. Device Error Matrix

Error	Direction	Packet Type	Data Buffer Error Bit	Transaction Error Bit
Data Buffer Overflow	RX	Any	1	0
ISO Packet Error	RX	ISO	0	1
ISO Fulfillment Error	Both	ISO	0	1

The device controller manages all errors on bulk/control/interrupt endpoints except for a data buffer overflow. However, for ISO endpoints, errors packets are not retried and errors are tagged as indicated.

Table 10-59. Error Descriptions

Overflow	Number of bytes received exceeded max. packet size or total buffer length. **Note:** This error also sets the halt bit in the dQH, and if there are dTDs remaining in the linked list for the endpoint, those are not executed.
ISO Packet Error	CRC error on received ISO packet. Contents not guaranteed correct.
ISO Fulfillment Error	Host failed to complete the number of packets defined in the dQH mult field within the given (micro)frame. For scheduled data delivery, DCD may need to readjust the data queue because a fulfillment error causes the device controller to cease data transfers on the pipe for one (micro)frame. During the dead (micro)frame, the device controller reports error on the pipe and primes for the following frame.

10.5.4 Servicing Interrupts

The interrupt service routine must understand there are high frequency, low frequency, and error operations to order accordingly.

10.5.4.1 High Frequency Interrupts

In particular, high frequency interrupts must be managed in the order below. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

Table 10-60. Interrupt Managing Order

Execution Order	Interrupt	Action
1a	USB Interrupt[1] EPSETUPSR	Copy contents of setup buffer and acknowledge setup packet (as indicated in Section 10.5.3.5, “Managing Queue Heads”). Process setup packet according to USB specification chapter 9 or application specific protocol.
1b	USB Interrupt EPCOMPLETE	Manage completion of dTD as indicated in Section 10.5.3.5, “Managing Queue Heads.”
2	SOF Interrupt	Action as deemed necessary by application. This interrupt may not have a use in all applications.

[1] It is likely multiple interrupts stack up on any call to the interrupt service routine and during interrupt service routine.

10.5.4.1.1 Low Frequency Interrupts

The low frequency events include the following interrupts. These interrupts can be managed in any order because they do not occur often in comparison to the high-frequency interrupts.

Table 10-61. Low Frequency Interrupt Events

Interrupt	Action
Port Change	Change software state information.
Sleep Enable (Suspend)	Change software state information. Low power managing as necessary.
Reset Received	Change software state information. Abort pending transfers.

10.5.4.1.2 Error Interrupts

Error interrupts are least frequent and should be placed last in the interrupt service routine.

Table 10-62. Error Interrupt Events

Interrupt	Action
USB Error Interrupt.	This error is redundant because it combines USB interrupt and an error status in the dTD. The DCD more aptly manages packet-level errors by checking the dTD status field upon receipt of USB interrupt (w/ EPCOMPLETE).
System Error	Unrecoverable error. Immediate reset of module; free transfers buffers in progress and restart the DCD.

10.5.5 Deviations from the EHCI Specifications

The host mode operation of the USB OTG module is nearly EHCI-compatible with a few minor differences. For the most part, the modules conform to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in the following areas:

- Embedded transaction translator—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation—In host mode, the device operational registers are generally disabled; therefore, device mode is mostly transparent when in host mode. However, there are a couple exceptions documented in the following sections.
- Embedded design interface—The module does not have a PCI Interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.

For the purposes of the USB OTG implementing a dual-role host/device controller with support for OTG applications, it is necessary to deviate from the EHCI specification. Device and OTG operation are not specified in the EHCI specification, and thus the implementation supported in the USB OTG module is proprietary.

10.5.5.1 Embedded Transaction Translator Function

The USB host mode supports directly connected full- and low-speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high-speed hub transaction translator. Although there is no separate transaction translator block in the system, the transaction translator function normally associated with a high-speed hub is implemented within the DMA and protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models existing in the EHCI specification to support full- and low-speed devices.

10.5.5.1.1 Capability Registers

These additions to the capability registers support the embedded Transaction translator function:

- N_TT added to HSCPARAMS - Host Controller Structural Parameters
- N_PTT added to HSCPARAMS - Host Controller Structural Parameters

See Section 10.3.3.3, “Host Controller Structural Parameters Register (HCSPARAMS)” for usage information.

10.5.5.1.2 Operational Registers

These additions to the operational registers support the embedded TT:

- Addition of the TTCTRL register.
- Addition of a two-bit port speed (PSPD) field to the PORTSC*n* register.

10.5.5.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a full-speed (FS) or low-speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable is set only in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a high-speed connection (chirp completes successfully).

The module always sets the port enable bit after the port reset operation regardless of the result of the host device chirp result, and the resulting port speed is indicated by the PORTSC*n*[PSPD] field. Therefore, the standard EHCI host controller driver requires an alteration to manage directly connected full- and low-speed devices or hubs. The change is a fundamental one summarized in Table 10-63.

Table 10-63. Functional Differences Between EHCI and EHCI with Embedded TT

Standard EHCI	EHCI with embedded Transaction Translator
After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS.	After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSC*n*.
FS and LS devices are assumed to be downstream from a HS hub. Therefore, all port-level control performs through the hub class to the nearest hub.	FS and LS device can be downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, port-level control acts using the hub class through the nearest hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSC*n*.
FS and LS devices are assumed to be downstream from a HS hub with HubAddr equal to X. [where HubAddr > 0 and HubAddr is the address of the hub where the bus transitions from HS to FS/LS (split target hub)]	FS and LS device can be downstream from a HS hub with HubAddr equal to X [HubAddr > 0] or directly attached [where HubAddr equals 0 and HubAddr is the address of the root hub where the bus transitions from HS to FS/LS (split target hub is the root hub)]

10.5.5.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the root hub. It is demonstrated here how hub address and endpoint speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS) – asynchronous (bulk/control endpoints) periodic (interrupt)

- Hub address equals 0
- Transactions to direct attached device/hub.

   - QH.EPS equals port speed
- Transactions to a device downstream from direct attached FS hub.
   - QH.EPS equals downstream device speed

NOTE

> When QH.EPS equals 01 (LS) and PORTSC*n*[PSPD] equals 00 (FS), a LS-pre-PID is sent before transmitting LS traffic.

Maximum packet size must equal 64 or less to prevent undefined behavior.

2. siTD (for direct attach FS) – Periodic (ISO endpoint)

- All FS ISO transactions:
   - Hub address equals 0
   - siTD.EPS equals 00 (full speed)

Maximum packet size must equal to 1023 or less to prevent undefined behavior.

10.5.5.1.5 Operational Model

The operational models are well defined for the behavior of the transaction translator (see USB 2.0 specification) and for the EHCI controller moving packets between system memory and a USB-HS hub. Because the embedded transaction translator exists within the USB host controller, no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections briefly discuss the operational model for how the EHCI and transaction translator operational models combine without the physical bus between. The following sections assume the reader is familiar with the EHCI and USB 2.0 transaction translator operational models.

Microframe Pipeline

The EHCI operational model uses the concept of H-frames and B-frames to describe the pipeline between the host (H) and the bus (B). The embedded transaction translator uses the same pipeline algorithms specified in the USB 2.0 specification for a hub-based transaction translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the microframe pipeline implemented in the embedded transaction translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 are ready to execute on the bus in B-frame 0.

When programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded transaction translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream hub-based transaction translators.

After periodic transfers are exhausted, any stored asynchronous transfer is moved. Asynchronous transfers are opportunistic because they execute when possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer cannot babble through the SOF (start of B-frame 0.)

Split State Machines

The start and complete-split operational model differs from EHCI ghtly because there is no bus medium between the EHCI controller and the embedded transaction translator. Where a start or complete-split operation would occur by requesting the split to the HS hub, the start/complete-split operation is simple an internal operation to the embedded transaction translator. Table 10-64 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 10-64. Emulated Handshakes

Condition	Emulate TT Response
Start-Split: All asynchronous buffers full	NAK
Start-Split: All periodic buffers full	ERR
Start-Split: Success for start of async. transaction	ACK
Start-Split: Start periodic transaction	No handshake (Ok)
Complete-Split: Failed to find transaction in queue	Bus time-out
Complete-Split: Transaction in queue is busy	NYET
Complete-Split: Transaction in queue is complete	Actual handshake from FS/LS device

Asynchronous Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded Transaction Translator:

- USB 2.0 – 11.17.3
  - Sequencing is provided and a packet length estimator ensures no full-/low-speed packet babbles into SOF time.
- USB 2.0 – 11.17.4
  - • Transaction tracking for 2 data pipes.
- USB 2.0 – 11.17.5
  - • Clear_TT_Buffer capability provided though the use of the TTCTRL register.

Periodic Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded transaction translator:

- USB 2.0 – 11.18.6.[1-2]
  - Abort of pending start-splits
    - EOF (and not started in microframes 6)
    - Idle for more than 4 microframes
  - Abort of pending complete-splits
    - EOF
    - Idle for more than 4 microframes
- USB 2.0 - 11.18.[7-8]
  - Transaction tracking for up to 4 data pipes.

  - No more than 4 periodic transactions (interrupt/isochronous) can be scheduled through the embedded TT per frame.
- Complete-split transaction searching.

NOTE

There is no data schedule mechanism for these transactions other than the microframe pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 ms) or else undefined behavior may result.

10.5.5.2 Device Operation

The co-existence of a device operational controller within the USB OTG module has little effect on EHCI compatibility for host operation. However, given that the USB OTG controller initializes in neither host nor device mode, the USBMODE register must be programmed for host operation before the EHCI host controller driver can begin EHCI host operations.

10.5.5.3 Non-Zero Fields in the Register File

Some of the reserved fields and reserved addresses in the capability registers and operational registers have use in device mode. Adhere to these steps:

- Write operations to all EHCI reserved fields (some of which are device fields in the USB OTG module) in the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the module must properly mask EHCI reserved fields (some of which are device fields in the USB OTG module registers).

10.5.5.4 SOF Interrupt

The SOF interrupt is a free running 125 µs interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. The free running interrupt is shared with the device mode start-of-frame interrupt. See Section 10.3.4.2, “USB Status Register (USBSTS),” and Section 10.3.4.3, “USB Interrupt Enable Register (USBINTR),” for more information.

10.5.5.5 Embedded Design

This is an embedded USB host controller as defined by the EHCI specification; therefore, it does not implement the PCI configuration registers.

10.5.5.5.1 Frame Adjust Register

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like those provided by the frame adjust register in the PCI configuration registers. Starts of microframes are timed precisely to 125 µs using the transceiver clock as a reference clock or a 60 Mhz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces.

10.5.5.6 Miscellaneous Variations from EHCI

10.5.5.6.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces that can operate in different modes when the module is configured with the software programmable physical interface modes. The control bits for selecting the PHY operating mode are added to the PORTSC*n* register providing a capability not defined by the EHCI specification.

10.5.5.6.2 Discovery

Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the PORTSC*n* register for a duration of 10 ms. Due to the complexity required to support the attachment of devices not high speed, a counter is present in the design that can count the 10 ms reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is summarized as:

- Port change interrupt—Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall set the PORTSC*n*[PR] bit to reset the device.
- Software shall clear the PORTSC*n*[PR] bit after 10 ms.
  - This step, necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a 0 to the reset bit while a reset is in progress, the write is ignored and the reset continues until completion.
- Port change interrupt—Port enable change occurs to notify the host controller that the device is now operational and at this point the port speed is determined.

Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation, which re-assigns the port owner for any device that does not connect at high speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner hand-off is not implemented. Therefore, PORTSC*n*[PO] bit is read-only and always reads 0.
- A 2-bit port speed indicator field has been added to PORTSC*n* to provide the current operating speed of the port to the host controller driver.
- A 1-bit high-speed indicator bit has been added to PORTSC*n* to signify that the port is in HS vs. FS/LS.
  - This information is redundant with the 2-bit port speed indicator field above.

Chapter 11
Chip Configuration Module (CCM)

11.1 Introduction

The chip-configuration module (CCM) controls the chip configuration for the device.

11.1.1 Block Diagram

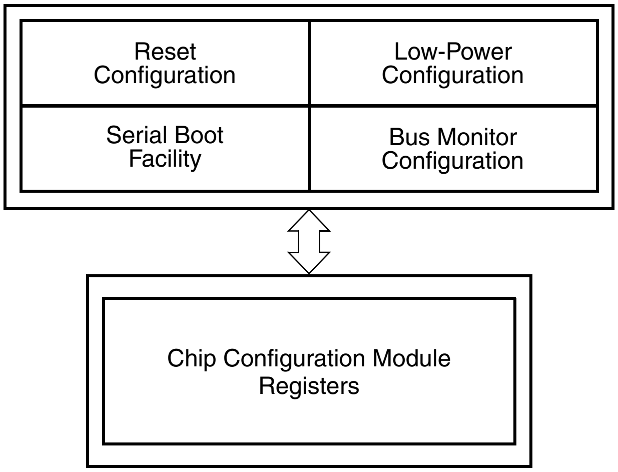


Figure 11-1. Chip-Configuration Module Block Diagram

11.1.2 Features

The CCM performs these operations:

- Configures device based on chosen reset configuration options
- Selects bus-monitor configuration
- Selects low-power configuration
- Facilitates serial boot (See Chapter 12, “Serial Boot Facility (SBF),” for details.)

11.1.3 Modes of Operation

The only chip operating mode available on this device is master mode. In master mode, the ColdFire core can access external memories and peripherals.The external bus consists of a 32-bit data bus and 24 address lines. The available bus control signals include R/$\overline{\text{W}}$, $\overline{\text{TS}}$, $\overline{\text{TA}}$, $\overline{\text{OE}}$, and $\overline{\text{BE/BWE}}$[3:0]. Up to four chip selects can be programmed to select and control external devices and to provide bus cycle termination.

11.2 External Signal Descriptions

Table 11-1 provides an overview of the CCM signals.

Table 11-1. Signal Properties

Name	Function	Reset State
BOOTMOD[1:0]	Reset configuration select	—
FB_AD[7:0]	Reset configuration override pins	—

11.2.1 BOOTMOD[1:0]

If the BOOTMOD[1:0] signals determine the boot performed at reset. See the table below for BOOTMOD[1:0] usage.

Table 11-2. BOOTMOD[1:0] Values

BOOTMOD[1:0]	Meaning
00	Boot from Flexbus with defaults.
01	Reserved.
10	Boot from Flexbus and override defaults via data bus (FB_AD[7:0]).
11	Boot from Flexbus and override defaults via the serial boot facility (SPI EEPROM/flash). See Chapter 12, “Serial Boot Facility (SBF).”

11.2.2 FB_AD[7:0] (Reset Configuration Override)

If the external BOOTMOD[1:0] pins are driven to 10 during reset, the states of the FB_AD[7:0] pins during reset determine Flexbus, PCI, and PLL configurations after reset.

NOTE

The logic levels for reset configuration on FB_AD[7:0] must be actively driven when BOOTMOD equals 10. FB_AD[31:8] should be allowed to float or be pulled high.

11.3 Memory Map/Register Definition

The CCM programming model consists of the registers listed in the below table.

Table 11-3. CCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers[1]					
0xFC0A_0004	Chip Configuration Register (CCR)	16	R	See Section	11.3.1/11-3
0xFC0A_0008	Reset Configuration Register (RCON)	16	R	0x03ED_0346	11.3.2/11-7
0xFC0A_000A	Chip Identification Register (CIR)	16	R	See Section	11.3.3/11-8

Table 11-3. CCM Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0010	Miscellaneous Control Register (MISCCR)	16	R/W	See Section	11.3.4/11-8
0xFC0A_0012	Clock Divider Register (CDR)	16	R/W	0x0001	11.3.5/11-11
0xFC0A_0014	USB On-the-Go Controller Status Register (UOCSR)	16	R/W	0x0010	11.3.6/11-11
0xFC0A_0018	Serial Boot Facility Status Register (SBFSR)[2]	16	R	See Section	12.3.1/12-3
0xFC0A_0020	Serial Boot Facility Control Register (SBFCR)[2]	16	R/W	See Section	12.3.2/12-3

[1] User access to supervisor-only address locations have no effect and result in a bus error.

[2] See Chapter 12, "Serial Boot Facility (SBF)," for details.

11.3.1 Chip Configuration Register (CCR)

The CCR is a read-only register; writing to the CCR has no effect. At reset, the CCR reflects the chosen operation of certain chip functions. These functions may be set to the defaults defined by the RCON register values or overridden during reset configuration using the external BOOTMOD[1:0] and either the FB_AD[7:0] pins or the serial boot data obtained from external SPI memory. (See Figure 11-4 for the RCON register definition.)

Three versions of the CCR are available, depending on the package type and the value of the CCR[FBCONFIG] bit field. These versions are shown below.

Address: 0xFC0A_0004 (CCR) Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R: FBCONFIG = 011, 111	0	0	0	0	0	0	1	1	FBCONFIG			PLL MODE	PCI MODE	PCI SLEW	PLLMULT	
R: FBCONFIG ≠ 011, 111	0	0	0	0	0	0	1	1	FBCONFIG			PLL MODE	OSC MODE	PLLMULT		
W																
Reset	0	0	0	0	0	0	1	1	See Note							

Note: Reset value depends upon chosen reset configuration. Default reset value (BOOTMOD = 00) is the value of RCON.

Figure 11-2. Chip Configuration Register (CCR) 360-pin

Table 11-4. CCR Field Descriptions 360-pin

Field	Description
15–10	Reserved, must be cleared.
9–8	Reserved, must be set.

Table 11-4. CCR Field Descriptions 360-pin (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–5
FBCONFIG</td><td>Flexbus, PCI, port size configuration. Relects the chosen Flexbus address/data muxing mode, PCI enable, and port size.
<table>
<tr><th>FBCONFIG</th><th>Flexbus A/D</th><th>PCI</th><th>Port Size</th></tr>
<tr><td>000</td><td>Non-muxed</td><td>Disabled</td><td>32-bit</td></tr>
<tr><td>001</td><td>Non-muxed</td><td>Disabled</td><td>8-bit</td></tr>
<tr><td>010</td><td>Non-muxed</td><td>Disabled</td><td>16-bit</td></tr>
<tr><td>011</td><td>Muxed</td><td>Enabled</td><td>16-bit</td></tr>
<tr><td>100</td><td>Muxed</td><td>Disabled</td><td>32-bit</td></tr>
<tr><td>101</td><td>Muxed</td><td>Disabled</td><td>8-bit</td></tr>
<tr><td>110</td><td>Muxed</td><td>Disabled</td><td>16-bit</td></tr>
<tr><td>111</td><td>Muxed</td><td>Enabled</td><td>8-bit</td></tr>
</table>
Muxed means that the FB_AD[31:0] signals are used for Flexbus address and data.
Non-muxed means that the FB_AD[31:0] signals are used for Flexbus data and the PCI_AD[31:0] signals are used for Flexbus address.
<b>Note:</b> The FBCONFIG field value may not be valid following serial boot, because serial boot reset configuration can select chip configurations not shown in the above table.</td></tr>
<tr><td>4
PLLMODE</td><td>PLL mode. Reflects the chosen overall clocking mode for the device.
0 Normal operation; PLL drives internal clocks
1 Limp mode; low-power clock divider drives internal clocks</td></tr>
<tr><td>3
PCIMODE
(FBCONFIG = 011, 111)</td><td>PCI host/agent mode, if the PCI is enabled. Reflects whether the PCI is a host or agent.
0 PCI is agent
1 PCI is host</td></tr>
<tr><td>3
OSCMODE
(FBCONFG ≠ 011, 111)</td><td>Oscillator clock mode, if the PCI is disabled.
0 Crystal oscillator mode
1 Oscillator bypass mode</td></tr>
</table>

Table 11-4. CCR Field Descriptions 360-pin (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
PCISLEW
(FBCONFG = 011, 111)</td><td>PCI pad slew rate mode, if the PCI is enabled. Reflects the slew rate on the PCI pads.
0 33 MHz slew rate
1 66 MHz slew rate</td></tr>
<tr><td>1–0
PLLMULT
(FBCONFG = 011, 111)

2–0
PLLMULT
(FBCONFG ≠ 011, 111)</td><td>PLL clock mode. Reflects the chosen PLL clock mode as set by the reference clock multiplier used to generate the VCO clock. The below table describes the PLLMULT settings for PCI-enabled 360-pin devices.
<table>
<tr><th>PLLMULT</th><th>VCO</th><th>CPU Frequency</th><th>PCI Frequency</th></tr>
<tr><td>00</td><td>12 × REF</td><td>200/240</td><td>33/40</td></tr>
<tr><td>01</td><td>6 × REF</td><td>200/180</td><td>66/50</td></tr>
<tr><td>10</td><td>16 × REF</td><td>266/200</td><td>33/25</td></tr>
<tr><td>11</td><td>8 × REF</td><td>266/200</td><td>66/50</td></tr>
</table>
<b>Note:</b> The PLLMULT field value may not be valid following serial boot, because serial boot reset configuration can select reference clock multipliers not shown directly above.
The below table describes the PLLMULT settings for PCI-disabled 360-pin devices.
<table>
<tr><th>PLLMULT</th><th>VCO</th></tr>
<tr><td>000</td><td>20 × REF</td></tr>
<tr><td>001</td><td>10 × REF</td></tr>
<tr><td>010</td><td>24 × REF</td></tr>
<tr><td>011</td><td>18 × REF</td></tr>
</table>
<table>
<tr><th>PLLMULT</th><th>VCO</th></tr>
<tr><td>100</td><td>12 × REF</td></tr>
<tr><td>101</td><td>6 × REF</td></tr>
<tr><td>110</td><td>16 × REF</td></tr>
<tr><td>111</td><td>8 × REF</td></tr>
</table>
<b>Note:</b> The PLLMULT field value may not be valid following serial boot, because serial boot reset configuration can select reference clock multipliers not shown in directly above.
The PLL output frequency can also be programmed after reset via the PLL output divider registers (PODR) and PLL feedback divider register (PFDR). See Chapter 8, “Clock Module,” for more details. The default output divider settings (values used to divide the VCO clock down to the system clocks) are shown in the below table.
<table>
<tr><th>Clock</th><th>PLL clock</th><th>OUTDIV Value</th><th>PODR</th></tr>
<tr><td>CPU</td><td>OUTDIV1</td><td>2</td><td>1</td></tr>
<tr><td>System bus</td><td>OUTDIV2</td><td>4</td><td>3</td></tr>
<tr><td>Flexbus (FB_CLK)</td><td>OUTDIV3</td><td>8</td><td>7</td></tr>
<tr><td>PCI clock</td><td>OUTDIV4</td><td>MULT[1]</td><td>MULT[1]-1</td></tr>
<tr><td>USB clock</td><td>OUTDIV5</td><td>8</td><td>7</td></tr>
</table>
[1] The value of MULT is the reference clock multiplier selected by the CCR[PLLMULT] field.</td></tr>
</table>

The CCR bit definition for 256-pin devices is shown in the below figure and table.

Address: 0xFC0A_0004 (CCR)

Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	1	1	FBCONFIG			PLL MODE	OSC MODE	PLLMULT		
W																
Reset	0	0	0	0	0	0	1	1	See Note							

Note: Reset value depends upon chosen reset configuration. Default reset value (BOOTMOD = 00) is the value of RCON.

Figure 11-3. Chip Configuration Register (CCR) 256-pin

Table 11-5. CCR Field Descriptions 256-pin

Field	Description
15–10	Reserved, must be cleared.
9–8	Reserved, must be set.
7–5 FBCONFIG	Flexbus, port size configuration. Relects the chosen Flexbus address/data muxing mode and port size. **FBCONFIG** | **Flexbus A/D** | **Port Size** 000 | Non-muxed | 32-bit 001 | Non-muxed | 8-bit 010 | Non-muxed | 16-bit 011 | Reserved 100 | Muxed | 32-bit 101 | Muxed | 8-bit 110 | Muxed | 16-bit 111 | Reserved Muxed means that the FB_AD[31:0] signals are used for Flexbus address and data. Non-muxed means that the FB_AD[31:0] signals are used for Flexbus data and the PCI_AD[23:0] signals are used for Flexbus address. **Note:** The FBCONFIG field value may not be valid following serial boot, because serial boot reset configuration can select chip configurations not shown in the above table.
4 PLLMODE	PLL mode. Reflects the chosen overall clocking mode for the device. 0 Normal operation; PLL drives internal clocks 1 Limp mode; low-power clock divider drives internal clocks

Table 11-5. CCR Field Descriptions 256-pin (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3
OSCMODE</td><td>Oscillator clock mode.
0 Crystal oscillator mode
1 Oscillator bypass mode</td></tr>
<tr><td>2–0
PLLMULT</td><td>PLL clock mode. Reflects the chosen PLL clock mode as set by the reference clock multiplier used to generate the VCO clock.
<table>
<tr><th>PLLMULT</th><th>VCO</th><th>PLLMULT</th><th>VCO</th></tr>
<tr><td>000</td><td>20 × REF</td><td>100</td><td>12 × REF</td></tr>
<tr><td>001</td><td>10 × REF</td><td>101</td><td>6 × REF</td></tr>
<tr><td>010</td><td>24 × REF</td><td>110</td><td>16 × REF</td></tr>
<tr><td>011</td><td>18 × REF</td><td>111</td><td>8 × REF</td></tr>
</table>
<b>Note:</b> The PLLMULT field value may not be valid following serial boot, because serial boot reset configuration can select reference clock multipliers not shown in the above table.
The PLL output frequency can also be programmed after reset via the PLL output divider registers (PODR) and PLL feedback divider register (PFDR). See Chapter 8, “Clock Module,” for more details. The default output divider settings (values used to divide the VCO clock down to the system clocks) are shown in the below table.
<table>
<tr><th>Clock</th><th>PLL clock</th><th>OUTDIV Value</th><th>PODR</th></tr>
<tr><td>CPU</td><td>OUTDIV1</td><td>2</td><td>1</td></tr>
<tr><td>System bus</td><td>OUTDIV2</td><td>4</td><td>3</td></tr>
<tr><td>FlexBus (FB_CLK)</td><td>OUTDIV3</td><td>8</td><td>7</td></tr>
<tr><td>USB clock</td><td>OUTDIV5</td><td>8</td><td>7</td></tr>
</table>
</td></tr>
</table>

11.3.2 Reset Configuration Register (RCON)

At reset, the RCON register determines the default operation of certain chip functions. All default functions defined by the RCON values can be overridden only during reset configuration (see Section 11.4.1, “Reset Configuration”) if the external BOOTMOD[1:0] pins are driven to 10 or 11. RCON is a read-only register and contains the same fields as the CCR register.

Two versions of the RCON are available, depending on package type. These versions are shown in Figure 11-4 and Figure 11-5. Only two versions are available, unlike three versions for the CCR, because there are only two sets of default values. Those default values make one of the three CCR versions (360-pin PCI-disabled) unavailable as a default configuration.

Address: 0xFC0A_0008 (RCON) Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	1	1	FBCONFIG			PLL MODE	PCI MODE	PCI SLEW	PLLMULT	
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	0	1	1	0	1

Figure 11-4. Reset Configuration Register (RCON) 360-pin

Address: 0xFC0A_0008 (RCON) Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	1	1	FBCONFIG			PLL MODE	OSC MODE	PLLMULT		
W																
Reset	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0

Figure 11-5. Reset Configuration Register (RCON) 256-pin

11.3.3 Chip Identification Register (CIR)

Address: 0xFC0A_000A (CIR) Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIN										PRN					
W																
Reset	Device Dependent										Mask Set Dependent					

Figure 11-6. Chip Identification Register (CIR)

Table 11-6. CIR Field Descriptions

Field	Description
15–6 PIN	Part identification number. Contains a unique identification number for the device. 0x04F MCF54450 0x04D MCF54451 0x04B MCF54452 0x049 MCF54453 0x04A MCF54454 0x048 MCF54455
5–0 PRN	Part revision number. This number increases by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order.

11.3.4 Miscellaneous Control Register (MISCCR)

The MISCCR register provides clock source selection and configuration for internal clocks, as well as SSI/timer DMA mux control and other miscellaneous control functionality.

Address: 0xFC0A_0010 (MISCCR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	LIMP	BME	BMT			SSI PUE	SSI PUS	TIM DMA	SSI SRC	0	USB PUD	USB OC	USB SRC
W																
Reset	0	0	0	See Note									0	0	See Note	

Note: Reset value depends on RCON type. See Table 11-7.

Figure 11-7. Miscellaneous Control Register (MISCCR)

Table 11-7. MISCCR Field Reset Values

Field	BOOTMOD[1:0]		
	00	10	11
LIMP	0	FB_AD4	SBF_RCON[111]
BME	1	1	SBF_RCON[123]
BMT	000	000	SBF_RCON[122:120]
SSIPUE	1	1	SBF_RCON[107]
SSIPUS	1	1	SBF_RCON[106]
TIMDMA	1	1	SBF_RCON[110]
SSISRC	1	1	SBF_RCON[105]
USBPUD	0	0	0
USBOC	1	1	SBF_RCON[108]
USBSRC	1	1	SBF_RCON[109]

Table 11-8. MISCCR Field Descriptions

Field	Description
15–13	Reserved, must be cleared.
12 LIMP	Limp mode enable. Selects between the PLL and the low-power clock divider as the source of all system clocks. 0 Normal operation; PLL drives system clocks. 1 Limp mode; low-power clock divider drives system clocks. **Note:** The transient behavior of the system when writing this bit cannot be predicted. When any USB wake-up event is detected, this bit is cleared, limp mode is exited, and the PLL begins the process of relocking and driving the system clocks.
11 BME	Bus monitor external enable bit. Enables the bus monitor to operate during external FlexBus cycles 0 Bus monitor disabled on external FlexBus cycles 1 Bus monitor enabled on external FlexBus cycles

Table 11-8. MISCCR Field Descriptions (continued)

Field	Description
10–8 BMT	Bus monitor timing field. Selects the timeout period in FlexBus clock cycles for the bus monitor: Timeout period for external bus cycles equals $2^{(16\text{-}BMT)}$ FB_CLK cycles 000 65536 001 32768 010 16384 011 8192 100 4096 101 2048 110 1024 111 512
7 SSIPUE	SSI RXD/TXD pull enable. Enables the internal weak pull cells on any external pin where either the SSI receive data (RXD) function or SSI transmit data (TXD) function is available. The affected pins include SSI_RXD, and SSI_TXD. 0 SSI data pin weak pull cells disabled. 1 SSI data pin weak pull cells enabled. **Note:** The SSIPUE bit enables only the pull cells when the SSI RXD and TXD functions are currently selected for the affected pins. See the Chapter 16, “Pin Multiplexing and Control,” for information on how to enable the SSI functions on those pins.
6 SSIPUS	SSI RXD/TXD pull select. Selects whether the internal weak pull cells enabled by the SSIPUE bit are pull up or pull down. 0 SSI data pins are pulled down. 1 SSI data pins are pulled up. **Note:** The SSIPUS bit has no effect when the SSIPUE bit is cleared.
5 TIMDMA	Timer DMA mux selection. Selects between the timer DMA signals and SSI DMA signals as those signals are mapped to DMA channels 9-12. Refer to the Chapter 19, “Enhanced Direct Memory Access (eDMA),” for more details on the DMA controller. 0 SSI RX0, SSI RX1, SSI TX0, and SSI TX1 DMA signals mapped to DMA channels 9 – 12, respectively. 1 Timer 0 – 3 DMA signals mapped to DMA channels 9 – 12, respectively.
4 SSISRC	SSI clock source. Selects between the PLL and the external SSI_CLKIN pin as the source of the SSI baud clock. 0 SSI_CLKIN pin directly drives SSI baud clock. 1 PLL drives SSI baud clock with fractionally divided CPU clock.
3	Reserved, must be cleared.
2 USBPUD	USB transceiver pull-up disable. Disables the USB OTG controller from driving the internal transceiver pull-up. 0 USB OTG drives the internal transceiver pull-up. 1 Internal transceiver pull-up is disabled. The USB_PULLUP signal is used to trigger the external pull-up.
1 USBOC	USB VBUS over-current sense polarity. Selects the polarity of the USB VBUS over-current sense signal driven off-chip. 0 USB_VBUS_OC is active high. 1 USB_VBUS_OC is active low.
0 USBSRC	USB clock source. Selects between the PLL and the external USB_CLKIN external pin as the clock source for the serial and ULPI interfaces of the USB module. 0 USB_CLKIN pin drives USB serial interface clocks. 1 PLL drives USB serial interface clocks.

11.3.5 Clock-Divider Register (CDR)

The CDR register provides clock division factors for limp mode and the SSI master clock when the PLL is used to drive the SSI clock.

Address: 0xFC0A_0012 (CDR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	LPDIV				SSIDIV							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 11-8. Clock-Divider Register (CDR)

Table 11-9. CDR Field Descriptions

Field	Description
15–12	Reserved, must be cleared.
11–8 LPDIV	Low power clock divider. Specifies the divide value used to produce the system clocks during limp mode. A 2:1 ratio is maintained between the core and the internal bus.This field is used only when MISCCR[LIMP] bit is set. $\text{System Clocks} = \frac{f_{\text{EXTAL}}}{2^{\text{LPDIV}}}$ ***Eqn. 11-1*** **Note:** When LPDIV is cleared (divide-by-1), the internal bus clock and FB_CLK do not have a 50/50 duty cycle.
7–0 SSIDIV	SSI oversampling clock divider. Specifies the divide value used to produce the oversampling clock for the SSI. This field is used only when MISCCR[SSISRC] is set (PLL is the source). $\text{SSI Baud Clock} = \frac{f_{\text{sys}}}{\text{SSIDIV}/2}$ ***Eqn. 11-2*** **Note:** A value of 0 or 1 for SSIDIV represents a divide-by-65. SSIDIV should not be set to any value that sets the SSI oversampling clock frequency over the bus clock frequency ($f_{sys/2}$), because incorrect SSI operation could result.

11.3.6 USB On-the-Go Controller Status Register (UOCSR)

The UOCSR register controls and reflects various features of the USB OTG module. When any bit of this register generates an interrupt, that interrupt can be cleared by reading the UOCSR register. The read-only bits of this register are set by the USB OTG module.

Address: 0xFC0A_0014 (UOCSR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	DPPD	DMPD	0	CRG_ VBUS	DCR_ VBUS	DPPU	AVLD	BVLD	VVLD	SEND	PWR FLT	WKUP	UOMIE	XPDE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 11-9. USB On-the-Go Controller Status Register (UOCSR)

Table 11-10. UOCSR Field Descriptions

Field	Description
15–14	Reserved, must be cleared.
13 DPPD	D+ 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D+ line is active. When set, asserts an interrupt if the UOMIE bit is set.
12 DMPD	D- 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D- line is active. When set, asserts an interrupt if the UOMIE bit is set.
11	Reserved, must be cleared.
10 CRG_VBUS	Charge VBUS. Indicates a charge resistor to pull-up VBUS is enabled. When set, asserts an interrupt if the UOMIE bit is set.
9 DCR_VBUS	Discharge VBUS. Indicates a discharge resistor to pull-down VBUS is enabled. When set, asserts an interrupt if the UOMIE bit is set.
8 DPPU	D+ pull-up control. Indicates pull-up on D+ for FS-only applications is enabled. When set, asserts an interrupt if the UOMIE bit is set.
7 AVLD	A-peripheral is valid. Indicates if the session for an A-peripheral is valid. 0 Session is not valid for an A-peripheral. 1 Session is valid for an A-peripheral.
6 BVLD	B-peripheral is valid. Indicates if the session for a B-peripheral is valid. 0 Session is not valid for a B-peripheral. 1 Session is valid for a B-peripheral.
5 VVLD	VBUS valid. Indicates if voltage on VBUS is at a valid level for operation. 0 Voltage level on VBUS is not valid for operation. 1 Voltage level on VBUS is valid for operation.
4 SEND	Session end. Indicates if voltage on VBUS has dropped below the session end threshold. 0 Voltage on VBUS has not dropped below the session end threshold 1 Voltage on VBUS has dropped below the session end threshold
3 PWRFLT	VBUS power fault. Indicates a power fault has occurred on VBUS (e.g. overcurrent). 0 No power fault has occurred. 1 Power fault has occurred.
2 WKUP	USB OTG controller wake-up event. Reflects if a wake-up event has occurred on the USB OTG controller bus 0 No outstanding wake-up event. 1 Wake-up event has occurred.
1 UOMIE	USB OTG miscellaneous interrupt enable. Enables an interrupt to generate from any of the following UOCSR bits: DPPD, DMPD, CRG_VBUS, DCR_VBUS, DPPU, and WKUP 0 Interrupt sources are disabled. 1 Interrupt sources are enabled.
0 XPDE	On-chip transceiver pull-down enable. 0 50 kΩ pull-downs disabled on OTG D+ and D- pins of on-chip transceiver. 1 On-chip 50 kΩ pull-downs enabled on OTG D+ and D- transceiver pins of on-chip transceiver.

11.4 Functional Description

11.4.1 Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states. Table 11-11 shows the states of the external pins while in reset.

Table 11-11. Reset Configuration Pin States During Reset

Pin	Pin Function	I/O	Input State
BOOTMOD[1:0]	BOOTMOD function for all modes	Input	Must be driven by external logic
FB_AD[7:0]	Flexbus address/data functions (BOOTMO does not equal 10)	Input	N/A
	Reset configuration data functions (BOOTMOD equals 10)	Input	Must be driven by external logic

11.4.1.1 Reset Configuration (BOOTMOD[1:0] = 00)

If the BOOTMOD pins are 00 during reset, the RCON register determines the chip configuration after reset, regardless of the states of the external data pins. The internal configuration signals are driven to levels specified by the RCON register's reset state for default module configuration.

11.4.1.2 Reset Configuration (BOOTMOD[1:0] = 10)

If the BOOTMOD pins are 10 during reset, the chip configuration after reset is determined according to the levels driven onto the FB_AD[7:0] pins. (See Table 11-12.) The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

NOTE

The logic levels for reset configuration on FB_AD[7:0] must be actively driven when BOOTMOD equals 10. The FB_AD[15:8] pins must be allowed to float or be pulled high.

Table 11-12. Parallel Configuration During Reset[1]

Pin(s) Affected	Default Configuration	Override Pins in Reset[2,3]	Function
FB_AD[31:0], PCI_*	See RCON[7:5]	**FB_AD[7:5]**	**Flexbus, PCI, Port Size Mode (360-pin Devices)**
		111	PCI, muxed FB addr/data, 8-bit boot
		110	No PCI, muxed FB addr/data, 16-bit boot
		101	No PCI, muxed FB addr/data, 8-bit boot
		100	No PCI, muxed FB addr/data, 32-bit boot
		011	PCI, muxed FB addr/data, 16-bit boot
		010	No PCI, non-muxed FB addr/data, 16-bit boot
		001	No PCI, non-muxed FB addr/data, 8-bit boot
		000	No PCI, non-muxed FB addr/data, 32-bit boot
FB_AD[31:0], PCI_*	See RCON[7:5]	**FB_AD[7:5]**	**Flexbus, PCI, Port Size Mode (256-pin Devices)**
		111	Reserved
		110	No PCI, muxed FB addr/data, 16-bit boot
		101	No PCI, muxed FB addr/data, 8-bit boot
		100	No PCI, muxed FB addr/data, 32-bit boot
		011	Reserved
		010	No PCI, non-muxed FB addr/data, 16-bit boot
		001	No PCI, non-muxed FB addr/data, 8-bit boot
		000	No PCI, non-muxed FB addr/data, 32-bit boot
(none)	See RCON[4]	**FB_AD4**	**PLL Mode**
		1	Limp mode
		0	PLL mode
(none)	See RCON[3]	**FB_AD3**	**PCI Host/Agent Mode (360-pin PCI-Enabled Devices)**
		1	PCI host mode
		0	PCI agent mode
(none)	See RCON[3]	**FB_AD3**	**Oscillator Mode (360-pin PCI-Disabled Devices and 256-pin Devices)**
		1	Oscillator bypass mode
		0	Crystal oscillator mode

Table 11-12. Parallel Configuration During Reset[1] (continued)

Pin(s) Affected	Default Configuration	Override Pins in Reset[2,3]	Function
PCI_*	See RCON[2]	**FB_AD2**	**PCI Slew Rate Mode (360-pin PCI-Enabled Devices)**
		1	66 MHz slew rate mode
		0	33 MHz slew rate mode
(none)	See RCON[2:0]	**FB_AD[2:0]**	**PLL Multiplier (360-pin PCI-Disabled Devices and 256-pin Devices)**
		111	VCO = 8 x REF
		110	VCO = 16 x REF
		101	VCO = 6 x REF
		100	VCO = 12 x REF
		011	VCO = 18 x REF
		010	VCO = 24 x REF
		001	VCO = 10 x REF
		000	VCO = 20 x REF
(none)	See RCON[1:0]	**FB_AD[1:0]**	**PLL Multiplier (360-pin PCI-Enabled Devices)**

		11	VCO = 8 x REF
			CPU = 266/200; PCI = 66/50

		10	VCO = 16 x REF
			CPU = 266/200; PCI = 33/25

		01	VCO = 6 x REF
			CPU = 200/180; PCI = 66/50

		00	VCO = 12 x REF
			CPU = 200/240; PCI = 33/40

[1] Modifying the default configurations through the FB_AD[7:0] pins is possible only if the external BOOTMOD[1:0] pins are 10 while $\overline{\text{RSTOUT}}$ is asserted.

[2] The FB_AD[31:8] pins do not affect reset configuration.

[3] The external reset override circuitry drives the data bus pins with the override values while $\overline{\text{RSTOUT}}$ is asserted. It must stop driving the data bus pins within one FB_CLK cycle after $\overline{\text{RSTOUT}}$ is negated. To prevent contention with the external reset override circuitry, the reset override pins are forced to inputs during reset and do not become outputs until at least one FB_CLK cycle after $\overline{\text{RSTOUT}}$ is negated.

11.4.1.3 Reset Configuration (BOOTMOD[1:0] = 11)

If the BOOTMOD pins are 11 during reset, the chip configuration after reset is determined by data obtained from external SPI memory through serial boot using the SBF_DI, SBF_DO, $\overline{\text{SBF\_CS}}$, and SBF_CK signals. The internal configuration signals are driven to reflect the data being received from the external SPI memory to allow for module configuration. See Chapter 12, “Serial Boot Facility (SBF),” for more details on serial boot.

NOTE

The serial configuration depends on the package of the device. Use Table 11-13 for 360-pin devices and Table 11-14 for 256-pin devices

Table 11-13. Serial Configuration During Reset for 360-pin Devices

Pin(s) Affected	Default Configuration	Override Serial RCON Bits	Function
(none)	See RCON[7:5]	**SBF_RCON[127:126]**	**Boot Port Size**
		11	0-bit port (FB_AD[31:0] configured for GPIO)
		10	8-bit port
		01	16-bit port
		00	32-bit port
PCI_AD[31:0] (360-pin)	See RCON[7:5]	**SBF_RCON[125]**	**PCI and Flexbus A/D Pin Mode**
		1	PCI disabled Flexbus non-muxed address/data mode
		0	PCI enabled Flexbus muxed address/data mode
(none)	See RCON[3]	**SBF_RCON[124]**	**Oscillator Mode**
		1	Oscillator bypass mode
		0	Crystal oscillator mode
(none)	See MISCCR[11]	**SBF_RCON[123]**	**Bus Monitor Enable**
		1	Bus monitor enabled
		0	Bus monitor disabled
(none)	See MISCCR[10:8]	**SBF_RCON[122:120]**	**Bus Monitor Timeout (FB_CLK Cycles)**
		000	65536
		001	32768
		010	16384
		011	8192
		100	4096
		101	2048
		110	1024
		111	512

Table 11-13. Serial Configuration During Reset for 360-pin Devices (continued)

Pin(s) Affected	Default Configuration	Override Serial RCON Bits	Function
(none)	24 (dec)	**SBF_RCON[119:112]**	**PLL Reference Clock Multiplier** This value is loaded into the PLL's PCR[PFDR].
(none)	See RCON[4]	**SBF_RCON[111]**	**PLL Mode**
		1	Limp mode
		0	PLL mode
(none)	See MISCCR[5]	**SBF_RCON[110]**	**Timer/SSI DMA Channel Mux Select**
		1	Timer 0-3 DMA signals mapped to DMA channels 9-12, respectively
		0	SSI RX0, SSI RX1, SSI TX0, SSI TX1 DMA signals mapped to DMA channels 9-12, respectively
(none)	See MISCCR[0]	**SBF_RCON[109]**	**USB Clock Source**
		1	PLL drives USB serial interface clocks
		0	USB_CLKIN pin drives USB serial interface clock
(none)	See MISCCR[1]	**SBF_RCON[108]**	**USB VBUS Overcurrent Sense Polarity**
		1	USB_VBUS_OC is active-high
		0	USB_VBUS_OC is active-low
(none)	See MISCCR[7]	**SBF_RCON[107]**	**SSI RXD/TXD Pull Enable**
		1	SSI_RXD,SSI_TXD pull cells enabled
		0	SSI_RXD,SSI_TXD pull cells disabled
(none)	See MISCCR[6]	**SBF_RCON[106]**	**SSI RXD/TXD Pull Select**
		1	SSI_RXD,SSI_TXD pulled high
		0	SSI_RXD,SSI_TXD pulled low
(none)	See MISCCR[4]	**SBF_RCON[105]**	**SSI Clock Source**
		1	PLL drives SSI clock
		0	SSI_CLKIN pin drives SSI clock
(none)	See RCON[2]	**SBF_RCON[104]**	**PCI Pad Slew Rate Mode**
		1	66 MHz slew rate mode
		0	33 MHz slew rate mode
(none)	See RCON[3] - host mode disables interrupt	**SBF_RCON[103]**	**PCI Interrupt**
		1	PCI interrupt enabled
		0	PCI interrupt disabled

Table 11-13. Serial Configuration During Reset for 360-pin Devices (continued)

Pin(s) Affected	Default Configuration	Override Serial RCON Bits	Function
(none)	See RCON[3] - host mode disables retry	**SBF_RCON[102]**	**PCI Configuration Retry**
		1	PCI configuration retry enabled
		0	PCI configuration retry disabled
(none)	See RCON[3] - host mode enables BAR	**SBF_RCON[101]**	**PCI BAR5 Enable**
		1	BAR5 enabled
		0	BAR5 disabled
(none)	See RCON[3] - host mode enables BAR	**SBF_RCON[100]**	**PCI BAR4 Enable**
		1	BAR4 enabled
		0	BAR4 disabled
(none)	See RCON[3] - host mode enables BAR	**SBF_RCON[99]**	**PCI BAR3 Enable**
		1	BAR3 enabled
		0	BAR3 disabled
(none)	See RCON[3] - host mode enables BAR	**SBF_RCON[98]**	**PCI BAR2 Enable**
		1	BAR2 enabled
		0	BAR2 disabled
(none)	See RCON[3] - host mode enables BAR	**SBF_RCON[97]**	**PCI BAR1 Enable**
		1	BAR1 enabled
		0	BAR1 disabled
(none)	See RCON[3] - host mode enables BAR	**SBF_RCON[96]**	**PCI BAR0 Enable**
		1	BAR0 enabled
		0	BAR0 disabled
(none)	$5807	**SBF_RCON[95:80]**	**PCI Device ID**
(none)	$1957	**SBF_RCON[79:64]**	**PCI Vendor ID**
(none)	$068000	**SBF_RCON[63:40]**	**PCI Class Code**
(none)	$00	**SBF_RCON[39:32]**	**PCI Revision ID**
(none)	$0000	**SBF_RCON[31:16]**	**PCI Subsystem ID**
(none)	$0000	**SBF_RCON[15:0]**	**PCI Subsystem Vendor ID**

Table 11-14. Serial Configuration During Reset for 256-pin Devices

Pin(s) Affected	Default Configuration	Override Serial RCON Bits	Function
(none)	See RCON[7:5]	**SBF_RCON[127:126]**	**Boot Port Size**
		11	0-bit port (FB_AD[31:0] configured for GPIO)
		10	8-bit port
		01	16-bit port
		00	32-bit port
PCI_AD[23:0]	See RCON[7:5]	**SBF_RCON[125]**	**Flexbus A/D Pin Mode**
		1	Flexbus non-muxed address/data mode
		0	Flexbus muxed address/data mode
(none)	See RCON[3]	**SBF_RCON[124]**	**Oscillator Mode**
		1	Oscillator bypass mode
		0	Crystal oscillator mode
(none)	See MISCCR[11]	**SBF_RCON[123]**	**Bus Monitor Enable**
		1	Bus monitor enabled
		0	Bus monitor disabled
(none)	See MISCCR[10:8]	**SBF_RCON[122:120]**	**Bus Monitor Timeout (FB_CLK Cycles)**
		000	65536
		001	32768
		010	16384
		011	8192
		100	4096
		101	2048
		110	1024
		111	512
(none)	24 (dec)	**SBF_RCON[119:112]**	**PLL Reference Clock Multiplier** This value is loaded into the PLL’s PCR[PFDR].
(none)	—	**SBF_RCON[111:96]**	**Must be Zero**

11.4.2 Boot Configuration

During reset configuration, the $\overline{\text{FB\_CS0}}$ chip select pin is always configured to select an external boot device. The valid (V) bit in the CSMR0 register is ignored and $\overline{\text{FB\_CS0}}$ is enabled after reset. $\overline{\text{FB\_CS0}}$ is asserted for the initial boot fetch accessed from address 0x0000_0000 for the stack pointer and address

0x0000_0004 for the program counter (PC). It is assumed the reset vector loaded from address 0x0000_0004 causes the processor to start executing from external memory space decoded by $\overline{\text{FB\_CS0}}$.

11.4.3 Low Power Configuration

After reset, the device can be configured for operation during the low power modes using the low power control register (LPCR). For more information on this register, see Section 9.2.5, “Low-Power Control Register (LPCR).”

Chapter 12
Serial Boot Facility (SBF)

12.1 Introduction

It is nearly impossible to dedicate and very impractical to share pins for the numerous available power-up options on today's complex, highly-integrated processors. The serial boot facility (SBF), shown in Figure 12-1, solves this problem by providing the user with the capability to store and load all device reset configuration data and user code from an external SPI memory. This method requires only a minimal number of I/O pins.

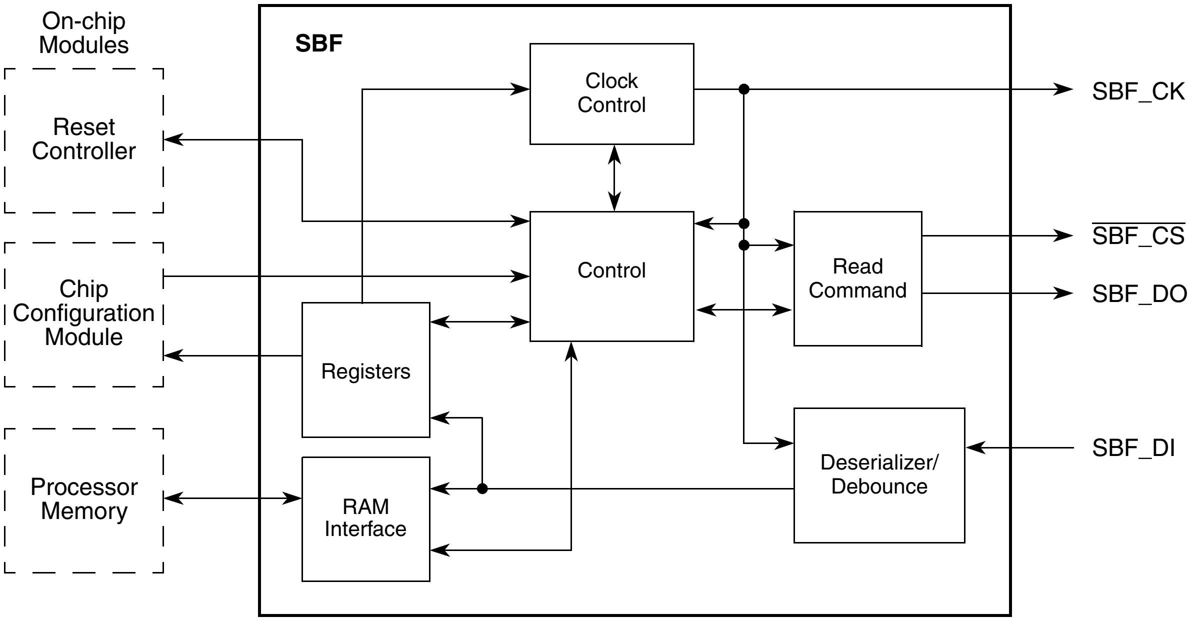


Figure 12-1. SBF Block Diagram

12.1.1 Overview

The SBF interfaces to an external SPI memory to read configuration data and boot code during the processor reset sequence if BOOTMOD[1:0] equals 11. By reading data stored in the SPI memory, the SBF adjusts the SPI memory clock frequency, configures an extended set of power-up options for the processor, and optionally loads code into the on-chip SRAM. Through interaction with the reset controller, the SBF performs these actions so that the chip is properly configured after exiting the reset state.

12.1.2 Features

The SBF includes these distinctive features:

- Support for many different SPI memory devices
  - — EEPROM
  - — Flash
  - — FRAM
  - — Embedded FPGA memory
- External interface maps directly to (and can be multiplexed with) the DMA serial peripheral interface (DSPI) pins
- Self-adjusting shift clock frequency for maximum throughput supported by SPI memory
- Optionally load boot code into processor’s memory space

12.2 External Signal Description

Listed below are the SBF module external signals.

Table 12-1. Signal Properties

Signal	I/O	Description	Reset	Pull Up
SBF_CK	O	Shift clock. Alternate edges of this signal cause the SPI memory to accept data from and drive data to the processor	—	—
$\overline{\text{SBF\_CS}}$	O	Chip select. This signal enables the SPI memory and places it into an active state, ready to accept commands.	—	—
SBF_DI	I	Data in. The SPI memory drives and the processor accepts read data on this signal.	—	Active[1]
SBF_DO	O	Data out. The SBF drives the read command and address on this signal.	—	—

[1] Disabled by the SBF when the SPI memory begins shifting out data.

12.3 Memory Map/Register Definition

The SBF programming model consists of the registers listed below.

Table 12-2. SBF Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0018	Serial boot facility status register (SBFSR)	16	R	See Section	12.3.1/12-3
0xFC0A_0020	Serial boot facility control register (SBFCR)	16	R/W	See Section	12.3.2/12-3

12.3.1 Serial Boot Facility Status Register (SBFSR)

The read-only SBFSR register reflects the amount of boot code loaded through the external SPI memory.

Address: 0xFC0A_0018 (SBFSR)

Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BLL															
W																
Reset	See Note															

Note: Reset value is user-defined (loaded from SPI memory during serial boot following any reset type)

Figure 12-2. Serial Boot Facility Status Register (SBFSR)

Table 12-3. SBFSR Field Descriptions

Field	Description
15–0 BLL	Boot load length. Reflects the number of longwords of boot code loaded from external SPI memory during serial boot. No boot code was loaded if BLL equals 0x0000. Otherwise, BLL plus 1 longwords were loaded.

12.3.2 Serial Boot Facility Control Register (SBFCR)

The read-always/write-once SBFCR register controls SBF operation following subsequent warm resets.

Address: 0xFC0A_0020 (SBFCR)

Access: User read/write-once

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	FR	BLDIV			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0[1]	See Note			

[1] Reset value is 0 and is reset only by power-on reset (remains unchanged for other reset types)

Note: The reset value is loaded from SPI memory during serial boot following power-on reset. It remains unchanged for other reset types. Prior to this register being loaded from SPI memory, a divisor of 67 is used to begin the serial boot sequence.

Figure 12-3. Serial Boot Facility Control Register (SBFCR)

Table 12-4. SBFCR Field Descriptions

Field	Description
15–5	Reserved, must be cleared.
4 FR	Fast read. Determines whether the SBF uses the standard READ command or flash FAST_READ command on reboot following any reset other than power-on reset. Because this register is write-once, the application must write the value for this bit in the same write that the BLDIV field is written. Any subsequent writes to this field prior to a power-on reset event terminate without effect. 0 SBF uses the standard READ command 1 SBF uses the FAST_READ command

Table 12-4. SBFCR Field Descriptions (continued)

Field	Description
3–0 BLDIV	Boot loader clock divider. Determines the SBF clock (PLL input reference clock) divisor that generates the serial shift clock output on SBF_CK. Prior to the serial boot sequence, a divisor of 67 is used. During the serial boot sequence, this field is loaded with the value read from the SPI memory. The application may write to this register to change the divisor for any subsequent serial boot that follows a soft-reset condition. Because this register is write-once, the application must write the value for this field in the same write that the FR bit is written (regardless of the value written to the FR bit). Any subsequent writes to this field prior to a power-on reset event terminate without effect.

BLDIV	Ideal Divisor	Shift Clock		BLDIV	Ideal Divisor	Shift Clock	
		High Time (f_{ref} Ticks)	Low Time (f_{ref} Ticks)			High Time (f_{ref} Ticks)	Low Time (f_{ref} Ticks)
0000	1	Bypass	Bypass	1000	14	7	7
0001	2	1	1	1001	17	9	8
0010	3	2	1	1010	25	13	12
0011	4	2	2	1011	33	17	16
0100	5	3	2	1100	34	17	17
0101	7	4	3	1101	50	25	25
0110	10	5	5	1110	67	34	33
0111	13	7	6	1111	Reserved		

12.4 Functional Description

When enabled, the SBF inserts three additional steps into the normal system boot process:

- Serial initialization and shift clock frequency adjustment
- Reset configuration and optional boot load
- Execution transfer

12.4.1 Serial Initialization and Shift Clock Frequency Adjustment

The following sequence is followed during a serial boot sequence:

1. The SBF is engaged when BOOTMOD[1:0] = 11 concurrent with the release of a pending source of reset (power-on, software watchdog, $\overline{\text{RESET}}$ pin, etc.).
2. Boot-up is paused.
3. The weak internal pull-up on SBF_DI is enabled. This allows a 1-to-0 transition to register when the SPI memory output switches from high-impedance to logic 0.
4. The SBF shifts the standard SPI memory read command (0x03) followed by repeated 0x00 address bytes to the SPI memory at $f_{REF} \div 60$.

5. After the SPI memory accepts however many shift clock edges are necessary to respond to the READ command, it turns on its previously tri-stated output and begins driving the msb of the byte at address 0.
   Bits [7:4] of this byte must be 0000, so that the required 1-to-0 transition can be detected on SBF_DI to synchronize the SBF state machine. If bits [7:4] of this byte are not 0000, bits[3:0] are ignored, another byte is clocked out of the SPI memory (SBF_DO remains at logic 0), and the SBF state machine again tests for a 1-to-0 transition followed by four consecutive zero bits.
6. After the necessary 1-to-0 transition and reception of a byte with bits [7:4] equal to 0000, the SBF pauses and bits [3:0] of the received byte select a new shift clock divider according to Table 12-4.
7. The weak internal pull-up on SBF_DI is disabled.
8. The shift clock begins toggling at the new frequency, resuming the READ command already in progress.

NOTE

Shift clock frequency adjustment follows a power-on/hard reset only. After the new divisor is known, it is stored in the sticky SBFCR[BLDIV] field and used for subsequent soft resets. This speeds reboot for systems that do not benefit from the optional FAST_READ on soft reset feature (e.g., the SPI memory does not support FAST_READ, or the input reference clock does not exceed the maximum allowable frequency for the READ command).

12.4.2 Reset Configuration and Optional Boot Load

After the steps in Section 12.4.1, “Serial Initialization and Shift Clock Frequency Adjustment”, are executed, the following is performed to load configuration data and optional boot code.

1. Next, the SBF shifts two bytes (16 bits) out of the SPI memory that indicate how many longwords, if any, are to be read during the optional boot load sequence. These bytes are software-visible in the SBFSR[BLL] field.
2. The read operation continues with four longwords (128 bits) of reset configuration data (one longword in the 256-pin devices), formatted in the order presented in Section 11.4.1.3, “Reset Configuration (BOOTMOD[1:0] = 11)”.
3. At this point, the SBF determines whether or not to read boot code. If SBFSR[BLL] is non-zero, BLL plus one longwords (4 × (BLL + 1) bytes) are consecutively loaded into the SRAM.

NOTE

Although the SBF permits up to 65,536 longwords (262,144 bytes) to be loaded, the maximum practical number that can be read is limited by the size of the device’s internal SRAM (8192 longwords (32,768 bytes) for this device).

12.4.3 Execution Transfer

After boot load is complete or if no boot load is requested (SBFSR[BLL] = 0), the following steps complete the serial boot process:

1. The acquired configuration data is driven to the appropriate modules.
2. The system is released from reset.
3. The ColdFire processor initiates its normal reset vector fetch at address 0.
4. The actual memory that responds to the reset vector fetch depends on whether serial boot load is requested:
   - If SBFSR[BLL] is cleared, the reset vector fetch is handled by the FlexBus module, and whatever external memory is mapped at address 0, governed by the user-provided setting of RCON/CCR[FBCONFIG].
   - If SBFSR[BLL] is set, the reset vector and boot code are read from the on-chip SRAM. (The SBF enables the SRAM and maps it to address 0 via the RAMBAR before control of the processor is restored to the ColdFire core.) The reset vector (initial stack pointer and program counter) should point to locations in the on-chip SRAM, so that boot code can initialize the device and load the application software from the SPI memory or via some other mechanism (e.g. a hard disk drive connected to the ATAPI controller or a network server responding to a TFTP client).

12.5 Initialization Information

12.5.1 SPI Memory Initialization

The SBF requires that, prior to device power-up, the SPI memory is loaded with data organized according to Table 12-5 or Table 12-6, depending on the exact device used (256- or 360-pin). See Section 11.4.1.3, “Reset Configuration (BOOTMOD[1:0] = 11),” for the reset configuration (SBF_RCON) data definition.

Table 12-5. SPI Memory Organization (360-pin Devices)

Byte Address	Data Contents
0x0	{0000,BLDIV[3:0]}
0x1	BLL[7:0]
0x2	BLL[15:8]
0x3	RCON[7:0]
0x4	RCON[15:8]
...	...
0x12	RCON[127:120]
0x13[1]	CODE_BYTE_0[2]
0x14[1]	CODE_BYTE_1

Table 12-5. SPI Memory Organization (360-pin Devices) (continued)

Byte Address	Data Contents
...	...
0x12 + 4 × (BLL + 1)[1]	CODE_BYTE_[4 × (BLL + 1) - 1]

[1] This assumes SBFSR[PLL] is non-zero. If PLL is zero, the SBF does not access data at these addresses.

[2] Start of user code copied into the on-chip SRAM. CODE_BYTE_0–3 is the supervisor stack pointer (SP) when loading completes. CODE_BYTE_4–7 is the program counter (PC) when loading completes.

Table 12-6. SPI Memory Organization (256-pin Devices)

Byte Address	Data Contents
0x0	{0000,BLDIV[3:0]}
0x1	BLL[7:0]
0x2	BLL[15:8]
0x3	0x00
0x4	0x00
0x5	RCON[119:112]
0x6	RCON[127:120]
0x7[1]	CODE_BYTE_0[2]
0x8[1]	CODE_BYTE_1
...	...
0x6 + 4 × (BLL + 1)[1]	CODE_BYTE_[4 × (BLL + 1) - 1]

[1] This assumes SBFSR[BLL] is non-zero. If BLL is zero, the SBF does not access data at these addresses.

[2] Start of user code copied into the on-chip SRAM. CODE_BYTE_0–3 is the supervisor stack pointer (SP) when loading completes. CODE_BYTE_4–7 is the program counter (PC) when loading completes.

12.5.2 FAST_READ Feature Initialization

Many SPI flash memories implement a FAST_READ command that allows for a substantially higher shift-clock frequency. The SBF always uses the normal read command when coming out of a power-on/hard reset. However, when coming out of a soft reset, it is possible to use FAST_READ because the SBF machine state is not lost.

For this reason, the SBFCR[FR] sticky bit may be set, causing the FAST_READ command to be issued instead of the read command in the event of a soft reset. To enable the FAST_READ feature, set SBFCR[FR] in the same write that sets the SBFCR[BLDIV] field. The value written to SBFCR[BLDIV] should correspond to the frequency the SPI memory supports in FAST_READ mode. After a soft reset, SBFCR[BLDIV] is not overwritten with the BLDIV[3:0] value read from the SPI memory. Instead, the SBF uses the SBFCR[BLDIV] value to determine the SPI memory clock.

NOTE

The ability to use the FAST_READ command is limited by the SBF electrical specifications. Specifically, delays present throughout the system (including those between the SBF, the pin multiplexing logic, and the actual I/O pads) effectively limit the maximum frequency at which the SBF operates and can preclude use of the FAST_READ feature altogether. Even when the delays within the processor itself are minimized, the actual SPI memories may have similarly untenable electrical specifications (data input setup and output valid times).

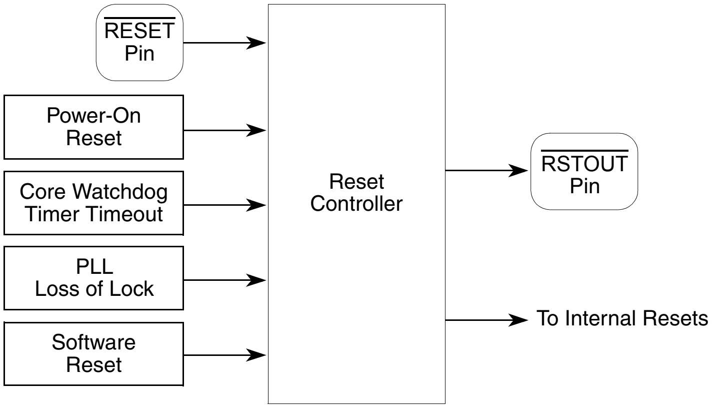
Chapter 13
Reset Controller Module

13.1 Introduction

The reset controller determines the cause of reset, asserts the appropriate reset signals to the system, and keeps a history of what caused the reset.

13.1.1 Block Diagram

Figure 13-1 illustrates the reset controller and is explained in these:



$\overline{\text{RESET}}$ Pin
Power-On Reset
Core Watchdog Timer Timeout
PLL Loss of Lock
Software Reset
Reset Controller
$\overline{\text{RSTOUT}}$ Pin
To Internal Resets

Figure 13-1. Reset Controller Block Diagram

13.1.2 Features

Module features include the following:

- Five sources of reset:
  - External
  - Power-on reset (POR)
  - Core watchdog timer
  - Phase locked-loop (PLL) loss of lock
  - Software
- Software-assertable $\overline{\text{RSTOUT}}$ pin independent of chip-reset state
- Software-readable status flags indicating the cause of the last reset

13.2 External Signal Description

Table 13-1 provides a summary of the reset-controller signal properties. The signals are described in the following paragraphs.

Table 13-1. Reset Controller Signal Properties

Name	I/O	Pull-up[1]	Input Hysteresis	Input Synchronization
$\overline{\text{RESET}}$	I	Active	Y	Y[2]
$\overline{\text{RSTOUT}}$	O	—	—	—

[1] All pull-ups are disconnected when the signal is programmed as an output.

[2] $\overline{\text{RESET}}$ is always synchronized except when in low-power stop mode.

13.2.1 $\overline{\text{RESET}}$

Asserting the external $\overline{\text{RESET}}$ for at least four rising FB_CLK edges causes the external reset request to be recognized and latched.

13.2.2 $\overline{\text{RSTOUT}}$

This active-low output signal is driven low when the internal reset controller module resets the device. It may take up to six FB_CLK edges after $\overline{\text{RESET}}$ assertion for $\overline{\text{RSTOUT}}$ to assert, due to an internal synchronizer on $\overline{\text{RESET}}$. When $\overline{\text{RSTOUT}}$ is active, the user can drive override options on the data bus. See Chapter 11, "Chip Configuration Module (CCM)," for more details on these override options.

13.3 Memory Map/Register Definition

The reset controller programming model consists of these registers:

- Reset control register (RCR), which selects reset control functions
- Reset status register (RSR), which reflects the state of the last reset source

See Table 13-2 for the memory map and the following paragraphs for register descriptions.

Table 13-2. Reset Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0000	Reset Control Register (RCR)	8	R/W	0x00	13.3.1/13-2
0xFC0A_0001	Reset Status Register (RSR)	8	R	See Section	13.3.2/13-3

13.3.1 Reset Control Register (RCR)

The RCR allows software control for requesting a reset and for independently asserting the external $\overline{\text{RSTOUT}}$ pin.

Address: 0xFC0A_0000 (RCR) Access: User read/write

	7	6	5	4	3	2	1	0
R	SOFTRST	FRCRSTOUT	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 13-2. Reset Control Register (RCR)

Table 13-3. RCR Field Descriptions

Field	Description
7 SOFTRST	Allows software to request a reset. The reset caused by setting this bit clears this bit. 1 Software reset request 0 No software reset request
6 FRCRSTOUT	Allows software to assert or negate the external $\overline{\text{RSTOUT}}$ pin. 1 Assert $\overline{\text{RSTOUT}}$ pin 0 Negate $\overline{\text{RSTOUT}}$ pin CAUTION: External logic driving reset configuration data during reset needs to be considered when asserting the $\overline{\text{RSTOUT}}$ pin when setting FRCRSTOUT.
5–0	Reserved, must be cleared.

13.3.2 Reset Status Register (RSR)

The RSR contains a status bit for every reset source. When reset is entered, the cause of the reset condition is latched, along with a value of 0 for the other reset sources that were not pending at the time of the reset condition. These values are then reflected in RSR. One or more status bits may be set at the same time. The cause of any subsequent reset is also recorded in the register, overwriting status from the previous reset condition.

RSR can be read at any time. Writing to RSR has no effect.

Address: 0xFC0A_0001 (RSR) Access: User read-only

	7	6	5	4	3	2	1	0
R	0	0	SOFT	0	POR	EXT	WDR CORE	LOL
W								
Reset:	0	0	Reset Dependent	0	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent

Figure 13-3. Reset Status Register (RSR)

Table 13-4. RSR Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5 SOFT	Software reset flag. Indicates the software caused last reset. 0 Last reset not caused by software 1 Last reset caused by software
4	Reserved, should be cleared.
3 POR	Power-on reset flag. Indicates power-on reset caused the last reset. 0 Last reset not caused by power-on reset 1 Last reset caused by power-on reset
2 EXT	External reset flag. Indicates that the last reset was caused by an external device or circuitry asserting the external $\overline{\text{RESET}}$ pin. 0 Last reset not caused by external reset 1 Last reset caused by external reset
1 WDRCORE	Core watchdog timer reset flag. Indicates the core watchdog timer timeout caused the last reset. 0 Last reset not caused by watchdog timer timeout 1 Last reset caused by watchdog timer timeout
0 LOL	Loss-of-lock reset flag. Indicates the last reset state was caused by a PLL loss of lock. 0 Last reset not caused by loss of lock 1 Last reset caused by a loss of lock

13.4 Functional Description

13.4.1 Reset Sources

Table 13-5 defines the reset sources and the signals driven by the reset controller.

Table 13-5. Reset Source Summary

Source	Type
Power on	Asynchronous
External $\overline{\text{RESET}}$ pin (not stop mode)	Synchronous
External $\overline{\text{RESET}}$ pin (during stop mode)	Asynchronous
Core Watchdog timer	Synchronous
Loss of lock	Synchronous
Software	Synchronous

To protect data integrity, a synchronous reset source is not acted upon by the reset control logic until the end of the current bus cycle. Reset is then asserted on the next rising edge of the system clock after the cycle is terminated. Internal byte, word, or longword writes are guaranteed to complete without data corruption when a synchronous reset occurs. External writes, including longword writes to 16-bit ports, are also guaranteed to complete.

Asynchronous reset sources usually indicate a catastrophic failure. Therefore, the reset control logic does not wait for the current bus cycle to complete. Reset is immediately asserted to the system.

13.4.1.1 Power-On Reset

At power up, the reset controller asserts $\overline{\text{RSTOUT}}$. $\overline{\text{RSTOUT}}$ continues to be asserted until V_{DD} has reached a minimum acceptable level and, if PLL clock mode is selected, until the PLL achieves phase lock. After approximately another 512 cycles (non-serial boot) or at the end of the serial boot sequence, $\overline{\text{RSTOUT}}$ is negated and the device begins operation.

13.4.1.2 External Reset

Asserting the external $\overline{\text{RESET}}$ for at least four rising FB_CLK edges causes the external reset request to be recognized and latched. After the $\overline{\text{RESET}}$ pin is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ either for approximately 512 bus clock cycles (non-serial boot) or for the duration of the serial boot sequence. The device then exits reset and begins operation.

In low-power stop mode, the system clocks stop. Asserting the external $\overline{\text{RESET}}$ in stop mode causes an external reset to be recognized asynchronously.

13.4.1.3 Core Watchdog Timer Reset

A core watchdog timer timeout causes the timer reset request to be recognized and latched. If the $\overline{\text{RESET}}$ pin is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ either for approximately 512 bus clock cycles (non-serial boot) or for the duration of the serial boot sequence. Then the device exits reset and begins operation.

13.4.1.4 Loss-of-Lock Reset

This reset condition occurs when the PLL loses lock. After the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ either for approximately 512 bus clock cycles (non-serial boot) or for the duration of the serial boot sequence. The device then exits reset and resumes operation.

13.4.1.5 Software Reset

A software reset occurs when the RCR[SOFTRST] bit is set. If the $\overline{\text{RESET}}$ is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ either for approximately 512 bus clock cycles (non-serial boot) or for the duration of the serial boot sequence. Then the device exits reset and resumes operation.

13.4.2 Reset Control Flow

The reset logic control flow is shown in Figure 13-4. In this figure, the control state boxes have been numbered, and these numbers are referred to (within parentheses) in the flow description that follows. All cycle counts given are approximate.

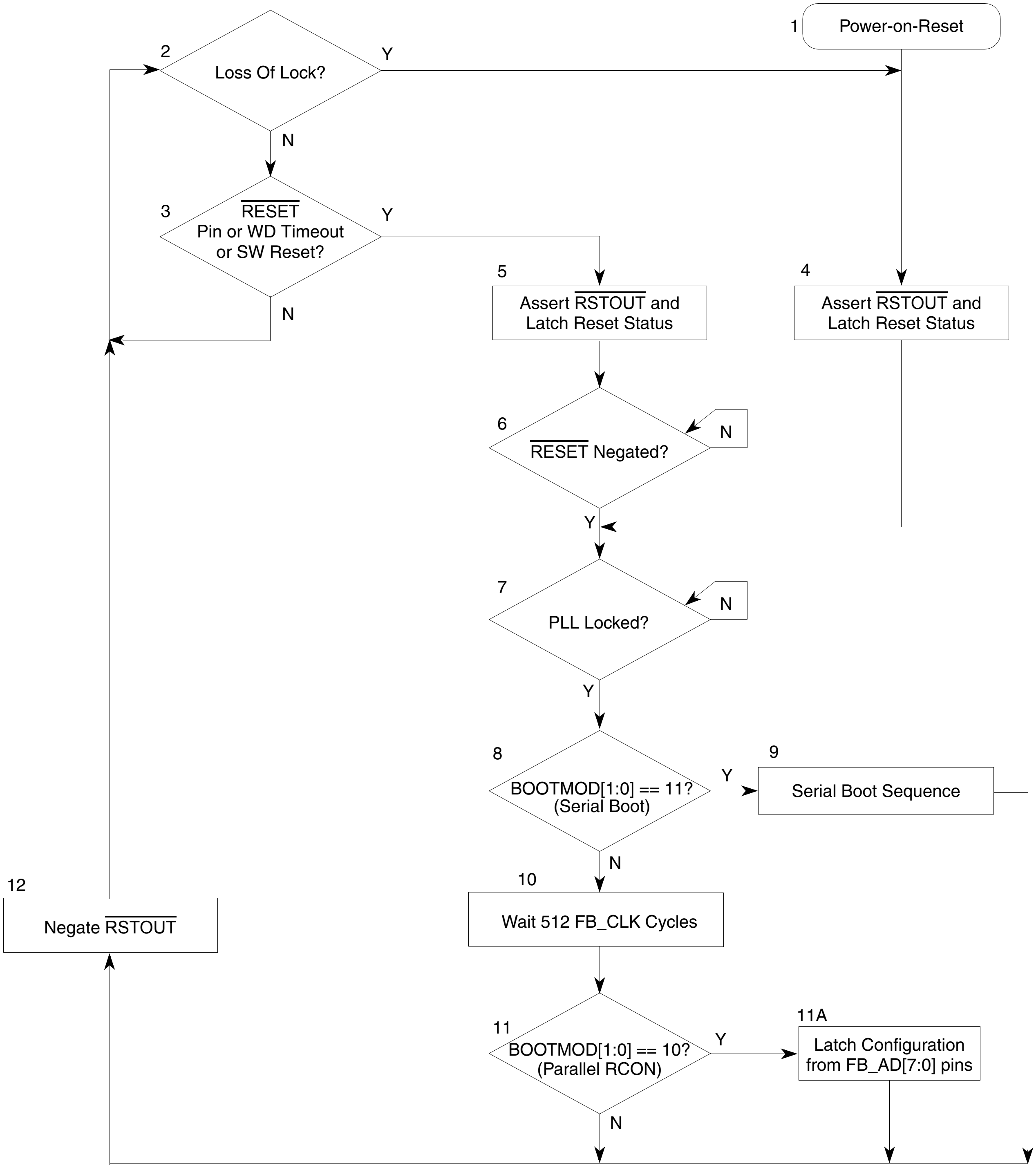


Figure 13-4. Reset Control Flow

13.4.2.1 Synchronous Reset Requests

In this discussion, the reference in parentheses refer to the state numbers in Figure 13-4. All cycle counts given are approximate.

If the external device asserts the external $\overline{\text{RESET}}$ signal for at least four rising FB_CLK edges (3), if the watchdog timer times out, or if software requests a reset, the reset control logic latches the reset request internally. At this point the $\overline{\text{RSTOUT}}$ pin is asserted (5). (Even though the external $\overline{\text{RESET}}$ pin needs to be asserted for only four FB_CLK edges, it may take up to six clocks beyond $\overline{\text{RESET}}$ assertion for $\overline{\text{RSTOUT}}$ to assert.) The reset control logic waits until the $\overline{\text{RESET}}$ signal is negated (6) and for the PLL to attain lock (7) before waiting 512 FB_CLK cycles (10) or for the duration of serial boot (9). For non-serial boot, the reset control logic may latch the chip configuration options from the FB_AD[7:0] pins (11, 11A). $\overline{\text{RSTOUT}}$ is then negated (12).

If the external $\overline{\text{RESET}}$ signal is asserted by an external device for at least four rising FB_CLK edges during the 512 count (10) or during the wait for PLL lock (7) or during serial boot (9), the reset flow switches to (6) and waits for the $\overline{\text{RESET}}$ signal to be negated before continuing.

13.4.2.2 Asynchronous Reset Request

If reset is asserted by an asynchronous internal reset source, such as loss of lock (2) or power-on reset (1), the reset control logic asserts $\overline{\text{RSTOUT}}$ (4). The reset control logic waits for the PLL to attain lock (7) before waiting either 512 bus clock cycles (10) or for the duration of serial boot (9). For non-serial boot, the reset control logic may then latch the chip configuration options from the FB_AD[7:0] pins (11, 11A). $\overline{\text{RSTOUT}}$ is then negated (12).

If a loss of lock occurs during the 512 bus clock count (10) or during serial boot (9), the reset flow switches to (7) and waits for the PLL to lock before continuing.

13.4.3 Concurrent Resets

This section describes the concurrent resets. As in the previous discussion references in parentheses refer to the state numbers in Figure 13-4.

13.4.3.1 Reset Flow

If a power-on reset is detected during any reset sequence, the reset sequence starts immediately (1).

If the external $\overline{\text{RESET}}$ pin is asserted for at least four rising FB_CLK edges while waiting for PLL lock or the 512 cycles or serial boot, the external reset is recognized. Reset processing switches to wait for the external $\overline{\text{RESET}}$ pin to negate (6).

If a loss-of-lock condition is detected during the 512 cycle wait or during serial boot, the reset sequence continues after a PLL lock (7).

13.4.3.2 Reset Status Flags

For a POR reset, the RSR[POR] bit is set, and all other RSR flags are cleared even if another type of reset condition is pending or concurrently asserted.

If other reset sources are asserted after the RSR status bits have been latched (4 or 5), the device is held in reset (9 or 10) until all sources have negated, and the subsequent sources are not reflected in the RSR contents.

Chapter 14
System Control Module (SCM)

14.1 Introduction

This system control module (SCM) provides several control functions, including peripheral access control, a software core watchdog timer, and generic access error information for the processor core.

14.1.1 Overview

The SCM provides programmable access protections for masters and peripherals. It allows the privilege level of a master to be overridden, forcing it to user-mode privilege, and allows masters to be designated as trusted or untrusted. Peripherals may be programmed to require supervisor privilege level for access, may restrict access to a trusted master only, and may be write-protected.

The SCM's core watchdog timer (CWT) provides a means of preventing system lockup due to uncontrolled software loops via a special software service sequence. If periodic software servicing action does not occur, the CWT times out with a programmed response (system reset or interrupt) to allow recovery or corrective action to be taken.

Fault access reporting is also available within the SCM. The user can use these registers during the resulting interrupt service routine and perform an appropriate recovery.

14.1.2 Features

The SCM includes these distinctive features:

- Access control registers
  - Master privilege register (MPR)
  - Peripheral access control registers (PACRs)
- System control registers
  - Core watchdog control register (CWCR) for watchdog timer control
  - Core watchdog service register (CWSR) to service watchdog timer
  - SCM interrupt status register (SCMISR) to service a bus fault or watchdog interrupt
  - Bus monitor timeout register (BMT)
- Core fault reporting registers

14.2 Memory Map/Register Definition

The memory map for the SCM registers is shown in Table 14-1.

Attempted accesses to reserved addresses result in a bus error, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, writes to the programming model must match the size of the register, e.g., an 8-bit register supports only 8-bit writes, etc. Attempted writes of a different size than the register width produce a bus error and no change to the targeted register.

Table 14-1. SCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_0000	Master Privilege Register (MPR)	32	R/W	0x7000_0007	14.2.1/14-2
0xFC00_0020	Peripheral Access Control Register A (PACRA)	32	R/W	0x5440_0000	14.2.2/14-4
0xFC00_0024	Peripheral Access Control Register B (PACRB)	32	R/W	0x0000_4404	14.2.2/14-4
0xFC00_0028	Peripheral Access Control Register C (PACRC)	32	R/W	0x4444_0444	14.2.2/14-4
0xFC00_002C	Peripheral Access Control Register D (PACRD)	32	R/W	0x4440_4444	14.2.2/14-4
0xFC00_0040	Peripheral Access Control Register E (PACRE)	32	R/W	0x4444_4444	14.2.2/14-4
0xFC00_0044	Peripheral Access Control Register F (PACRF)	32	R/W	0x4444_4444	14.2.2/14-4
0xFC00_0048	Peripheral Access Control Register G (PACRG)	32	R/W	0x4444_4444	14.2.2/14-4
0xFC04_0013	Wakeup Control Register (WCR)[1]	8	R/W	0x00	9.2.1/9-2
0xFC04_0016	Core Watchdog Control Register (CWCR)	16	R/W	0x0000	14.2.3/14-7
0xFC04_001B	Core Watchdog Service Register (CWSR)	8	R/W	Undefined	14.2.4/14-8
0xFC04_001F	SCM Interrupt Status Register (SCMISR)	8	R/W	0x00	14.2.5/14-9
0xFC04_0024	Burst Configuration Register (BCR)	32	R/W	0x0000_0000	14.2.6/14-10
0xFC04_0070	Core Fault Address Register (CFADR)	32	R	Undefined	14.2.7/14-10
0xFC04_0075	Core Fault Interrupt Enable Register (CFIER)	8	R/W	0x00	14.2.8/14-11
0xFC04_0076	Core Fault Location Register (CFLOC)	8	R	Undefined	14.2.9/14-11
0xFC04_0077	Core Fault Attributes Register (CFATR)	8	R	Undefined	14.2.10/14-12
0xFC04_007C	Core Fault Data Register (CFDTR)	32	R	Undefined	14.2.11/14-13

[1] The WCR register is described in Chapter 9, "Power Management."

14.2.1 Master Privilege Register (MPR)

The MPR specifies five 4-bit fields defining the access-privilege level associated with a bus master in the device to the various peripherals listed in Table 14-4. The register provides one field per bus master.

Address: 0xFC00_0000 (MPR)   Access: User 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
R	MPROT0				MPROT1				MPROT2				MPROT3				0	0	0	0	MPROT5				MPROT6				0	1	1	1
W																																
Reset	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 14-1. Master Privilege Register (MPR)

Each master is assigned depending on its connection to the various crossbar switch master ports.

Table 14-2. MPROT*n* Assignments

Crossbar Switch Port Number	MPROT*n*	Master
M0	MPROT0	ColdFire Core
M1	MPROT1	eDMA Controller
M2	MPROT2	FEC0
M3	MPROT3	FEC1
M5	MPROT5	PCI Controller
M6	MPROT6	USB On-the-Go
M7	MPROT7[1]	Serial Boot

[1] This field is located at MPR[3:0]. However, it is hardwired to 0111 and may not be altered.

The MPROT*n* field is defined as shown below.

	3	2	1	0
R	0	MTR	MTW	MPL
W				

Figure 14-2. MPROT*n* Fields

Table 14-3. MPROT*n* Field Descriptions

Field	Description
3	Reserved, must be cleared.
2 MTR	Master trusted for read. Determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
1 MTW	Master trusted for writes. Determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
0 MPL	Master privilege level. Determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode. 1 Accesses from this master are not forced to user-mode.

14.2.2 Peripheral Access Control Registers (PACR*x*)

Each of the peripherals has a four-bit PACR*n* field which defines the access levels supported by the given module. Eight PACRs are grouped together to form a 32-bit PACR*x* register (PACRA-PACRG). At reset, the SCM (PACR0) does not allow access from untrusted masters, while the other peripherals do.

Address: 0xFC00_0020 (PACRA) Access: User 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
R	PACR0				PACR1				PACR2				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-3. Peripheral Access Control Register A (PACRA)

Address: 0xFC00_0024 (PACRB) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PACR12				PACR13				0	0	0	0	PACR15			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 14-4. Peripheral Access Control Register B (PACRB)

Address: 0xFC00_0028 (PACRC) Access: User 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
R	PACR16				PACR17				PACR18				PACR19				0	0	0	0	PACR21				PACR22				PACR23			
W																																
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0

Figure 14-5. Peripheral Access Control Register C (PACRC)

Address: 0xFC00_002C (PACRD) Access: User 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
R	PACR24				PACR25				PACR26				0	0	0	0	PACR28				PACR29				PACR30				PACR31			
W																																
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0

Figure 14-6. Peripheral Access Control Register D (PACRD)

Address: 0xFC00_0040 (PACRE) Access: User 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
R	PACR32				PACR33				PACR34				PACR35				0	1	0	0	PACR37				0	1	0	0	0	1	0	0
W																																
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0

Figure 14-7. Peripheral Access Control Register E (PACRE)

Address: 0xFC00_0044 (PACRF) Access: User 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
R W	PACR40	PACR41	PACR42	PACR43	PACR44	PACR45	PACR46	PACR47
Reset	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0

Figure 14-8. Peripheral Access Control Register F (PACRF)

Address: 0xFC00_0048 (PACRG) Access: User 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
R	PACR48	PACR49	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0
W								
Reset	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0	0 1 0 0

Figure 14-9. Peripheral Access Control Register G (PACRG)

Each peripheral is assigned to its PACR*n* field:

Table 14-4. PACR*n* Assignments

Slot Number	PACR*n*	Peripheral
0	PACR0	SCM (MPR and PACRs)
1	PACR1	Crossbar switch
2	PACR2	FlexBus
12	PACR12	FEC0
13	PACR13	FEC1
15	PACR15	Real-Time Clock
16	PACR16	SCM (CWT and Core Fault Registers)
17	PACR17	eDMA Controller
18	PACR18	Interrupt Controller 0
19	PACR19	Interrupt Controller 1
21	PACR21	Interrupt Controller IACK
22	PACR22	I^2C
23	PACR23	DSPI
24	PACR24	UART0
25	PACR25	UART1
26	PACR26	UART2
28	PACR28	DMA Timer 0
29	PACR29	DMA Timer 1
30	PACR30	DMA Timer 2
31	PACR31	DMA Timer 3

Table 14-4. PACR*n* Assignments (continued)

Slot Number	PACR*n*	Peripheral
32	PACR32	PIT 0
33	PACR33	PIT 1
34	PACR34	PIT 2
35	PACR35	PIT 3
37	PACR37	Edge Port
40	PACR40	CCM, Reset Controller, Power Management
41	PACR41	Pin Multiplexing and Control (GPIO)
42	PACR42	PCI Controller
43	PACR43	PCI Arbiter
44	PACR44	USB On-the-Go
45	PACR45	RNG
46	PACR46	SDRAM Controller
47	PACR47	SSI
48	PACR48	ATA Controller
49	PACR49	PLL

The PACR*n* field is defined as:

	3	2	1	0
R	0	SP	WP	TP
W	▨			

Figure 14-10. PACR*n* Fields

Table 14-5. PACR*n* Field Descriptions

Field	Description
3	Reserved, must be cleared.
2 SP	Supervisor protect. Determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The master privilege level must indicate supervisor access attribute, and the MPROT*n*[MPL] control bit for the master must be set. If not, access terminates with an error response and no peripheral access initiates.

Table 14-5. PACR*n* Field Descriptions (continued)

Field	Description
1 WP	Write protect. Determines whether the peripheral allows write accesses 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, access terminates with an error response and no peripheral access initiates.
0 TP	Trusted protect. Determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access terminates with an error response and no peripheral access initiates.

14.2.3 Core Watchdog Control Register (CWCR)

The CWCR controls the software watchdog timer, time-out periods, and software watchdog timer interrupt. The register can be read or written at any time. At system reset, the software watchdog timer is disabled.

Address: 0xFC04_0016 (CWCR)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RO	0	0	0	0	0	0	CW RWH	CWE	CWRI		CWT				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-11. Core Watchdog Control Register (CWCR)

Table 14-6. CWCR Field Descriptions

Field	Description
15 RO	Read-only control bit. 0 CWCR can be read or written. 1 CWCR is read-only. A system reset is required to clear this register. The setting of this bit is intended to prevent accidental writes of the CWCR from changing the defined core watchdog configuration.
14–9	Reserved, must be cleared.
8 CWRWH	Core watchdog run while halted. 0 Core watchdog timer stops counting if the core is halted. 1 Core watchdog timer continues to count even while the core is halted.
7 CWE	Core watchdog timer enable. 0 CWT is disabled. 1 CWT is enabled.

Table 14-6. CWCR Field Descriptions (continued)

Field	Description
6–5 CWRI	Core watchdog reset/interrupt. 00 If a time-out occurs, the CWT generates an interrupt to the core. Refer to Chapter 17, “Interrupt Controller Modules,” for details on setting its priority level. 01 The first time-out generates an interrupt to the processor, and if not serviced, a second time-out generates a system reset and sets the RSR[WDRCORE] flag in the reset controller. 10 If a time-out occurs, the CWT generates a system reset and RSR[WDRCORE] in the reset controller is set. 11 The CWT functions in a window mode of operation. For this mode, the servicing of the CWSR must occur during the last 25% of the time-out period. Any writes to the CWSR during the first 75% of the time-out period generate an immediate system reset. Likewise, if the CWSR is not serviced during the last 25% of the time-out period, a system reset is generated. For any type of reset response, the RSR[WDRCORE] flag is set.
4–0 CWT	Core watchdog time-out period. Selects the time-out period for the CWT. At reset, this field is cleared selecting the minimum time-out period, but the CWT is disabled because CWCR[CWE] is cleared at reset. If CWCR[CWT] is *n*, the time-out period equals 2^n system clock cycles. However, if *n* is less than 8, the time-out period is forced to 2^8. 0x00 2^8 ... 0x08 2^8 0x09 2^9 ... 0x1F 2^{31}

14.2.4 Core Watchdog Service Register (CWSR)

The software watchdog service sequence must be performed using the CWSR as a data register to prevent a CWT time-out. The service sequence requires two writes to this data register: a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the CWT time-out, but any number of instructions can be executed between the two writes. If the CWT has already timed out, writing to this register has no effect in negating the CWT interrupt or reset. Figure 14-12 illustrates the CWSR. At system reset, the contents of CWSR are uninitialized.

NOTE

> If the CWT is enabled and has not timed out, any write of a data value other than 0x55 or 0xAA causes an immediate system reset, regardless of the value in the CWCR[CWRI] field.

Address: 0xFC04_001B (CWSR) Access: User read/write

	7	6	5	4	3	2	1	0
R W	CWSR							
Reset:	—	—	—	—	—	—	—	—

Figure 14-12. Core Watchdog Service Register (CWSR)

14.2.5 SCM Interrupt Status Register (SCMISR)

For certain values in the CWCR[CWRI] field, the CWT generates an interrupt response to a time-out. For these configurations, the SCMISR provides a program visible interrupt request from the watchdog timer. During the interrupt service routine which handles this interrupt, the source must be explicitly cleared by writing a 0x01 to the SCMISR.

The SCMISR also indicates system bus fault errors. An interrupt is sent only to the interrupt controller when the CFIER[ECFEI] bit is set. The SCMISR[CFEI] bit flags fault errors independent of the CFIER[ECFEI] setting. Therefore, if CFEI is set prior to setting ECFEI, an interrupt is requested immediately after ECFEI is set.

Address: 0xFC04_001F (SCMISR) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	CFEI	CWIC
W							w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 14-13. SCM Interrupt Status Register (SCMISR)

Table 14-7. SCMISR Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1 CFEI	Core fault error interrupt flag. Indicates if a bus fault has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect. 0 No bus error. 1 A bus error has occurred. The faulting address, attributes (and possibly write data) are captured in the CFADR, CFATR, and CFDTR registers. The error interrupt is enabled only if CFLOC[ECFEI] is set. **Note:** This bit reports core faults regardless of the setting of CFIER[ECFEI]. Therefore, if the error interrupt is disabled and a core fault occurs, this bit is set. Then, if the error interrupt is subsequently enabled, an interrupt is immediately requested. To prevent an undesired interrupt, clear the captured error by writing one to CFEI before enabling the interrupt.
0 CWIC	Core watchdog interrupt flag. Indicates whether an CWT interrupt has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect. 0 No CWT interrupt has occurred. 1 CWT interrupt has occurred.

14.2.6 Burst Configuration Register (BCR)

The BCR register enables or disables the USB On-the-Go module for bursting to/from the crossbar switch slave modules. There is an enable field for the slaves, and either direction (read and write) is supported via the GBR and GBW bits.

Address: 0xFC04_0024 (BCR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	GBR	GBW	SBE							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-14. Burst Configuration Register (BCR)

Table 14-8. BCR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9 GBR	Global burst enable for reads. Allows bursts to happen on read transactions from the crossbar switch slaves to the USB On-the-Go module. 0 Read bursts are disabled. 1 Read bursts are enabled. **Note:** If GBR and GBW are cleared, then SBE is ignored.
8 GBW	Global burst enable for writes. Allows bursts to happen on write transactions to the crossbar switch slaves from the USB On-the-Go module. 0 Write bursts are disabled. 1 Write bursts are enabled. **Note:** If GBR and GBW are cleared, then SBE is ignored.
7–0 SBE	Slave burst enable. Allows bursts to happen to/from the crossbar switch slaves. The only valid settings for this field are 0x00 or 0xFF. 0x00 Bursts disabled. 0xFF Bursts enabled. The GBR and GBW bits determine the burst direction. If neither is set, then this bit has no effect. Else Reserved.

14.2.7 Core Fault Address Register (CFADR)

The CFADR is a read-only register indicating the address of the last core access terminated with an error response.

Address: 0xFC04_0070 (CFADR) Access: User 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
R	ADDR																															
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 14-15. Core Fault Address Register (CFADR)

Table 14-9. CFADR Field Descriptions

Field	Description
31–0 ADDR	Indicates the faulting address of the last core access terminated with an error response.

14.2.8 Core Fault Interrupt Enable Register (CFIER)

The CFIER register enables the system bus-error interrupt. See Chapter 17, "Interrupt Controller Modules," for more information of the interrupt controller.

Address: 0xFC04_0075 (CFIER) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	ECFEI
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-16. Core Fault Interrupt Enable Register (CFIER)

Table 14-10. CFIER Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 ECFEI	Enable core fault error interrupt. 0 Do not generate an error interrupt on a faulted system bus cycle. 1 Generate an error interrupt to the interrupt controller on a faulted system bus cycle.

14.2.9 Core Fault Location Register (CFLOC)

The read-only CFLOC register indicates the exact location within the device of the captured fault information.

Address: 0xFC04_0076 (CFLOC) Access: User read-only

	7	6	5	4	3	2	1	0
R	LOC	0	0	0	0	0	0	0
W								
Reset:	–	0	0	0	0	0	0	0

Figure 14-17. Core Fault Location Register (CFLOC)

Table 14-11. CFLOC Field Descriptions

Field	Description
7 LOC	The location of the last captured fault. 0 Error occurred on the internal bus. 1 Error occurred within the core.
6–0	Reserved, must be cleared.

14.2.10 Core Fault Attributes Register (CFATR)

The read-only CFATR register captures the processor’s attributes of the last faulted core access to the system bus.

Address: 0xFC04_0077 (CFATR) Access: User read-only

	7	6	5	4	3	2	1	0
R	WRITE	SIZE			CACHE	0	MODE	TYPE
W								
Reset:	–	–	–	–	–	–	–	–

Figure 14-18. Core Fault Attributes Register (CFATR)

Table 14-12. CFATR Field Descriptions

Field	Description
7 WRITE	Indicates the direction of the last faulted core access. 0 Core read access. 1 Core write access.
6–4 SIZE	Indicates the size of the last faulted core access. 000 8-bit core access. 001 16-bit core access. 010 32-bit core access. Else Reserved.
3 CACHE	Indicates if last faulted core access was cacheable. 0 Non-cacheable 1 Cacheable
2	Reserved, must be cleared.
1 MODE	Indicates the mode the device was in during the last faulted core access. 0 User mode 1 Supervisor mode
0 TYPE	Defines the type of last faulted core access. 0 Instruction 1 Data

14.2.11 Core Fault Data Register (CFDTR)

The CFDTR is a read-only register for capturing the data associated with the last faulted processor write data access from the device's internal bus. The CFDTR is valid only for faulted internal bus-write accesses, CFLOC[LOC] is cleared.

Address: 0xFC04_007C (CFDTR) Access: User 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
R	CFDTR																															
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 14-19. Core Fault Data Register (CFDTR)

Table 14-13. CFDTR Field Descriptions

Field	Description
31–0 CFDTR	Contains data associated with the faulting access of the last internal bus write access. Contains the data value taken directly from the write data bus.

14.3 Functional Description

14.3.1 Access Control

The SCM supports the traditional model of two privilege levels: supervisor and user. Typically, memory references with the supervisor attribute have total accessibility to all the resources in the system, while user mode references cannot access system control and configuration registers. In many systems, the operating system executes in supervisor mode, while application software executes in user mode.

The SCM further partitions the access-control functions into two parts: one control register defines the privilege level associated with each bus master (MPR), and another set of control registers define the access levels associated with the peripheral modules (PACR*x*).

Each bus transaction targeted for the peripheral space is first checked to see if its privilege rights allow access to the given memory space. If the privilege rights are correct, the access proceeds on the internal bus. If the privilege rights are insufficient for the targeted memory space, the transfer is immediately aborted and terminated with an exception, and the targeted module not accessed.

14.3.2 Core Watchdog Timer

The core watchdog timer (CWT) prevents system lockup if the software becomes trapped in a loop with no controlled exit or if a bus transaction becomes hung. The core watchdog timer can be enabled through CWCR[CWE]; it is disabled at reset. If enabled, the CWT requires the periodic execution of a core watchdog servicing sequence. If this periodic servicing action does not occur, the timer expires and, depending on the setting of CWCR[CWRI], different events may occur:

- An interrupt may be generated to the core.
- An immediate system reset.

- Upon the first time-out, a watchdog timer interrupt is asserted. If this time-out condition is not serviced before a second time-out occurs, the CWT asserts a system reset. This configuration supports a more graceful response to watchdog time-outs.
- In addition to these three basic modes of operation, the CWT also supports a windowed mode of operation. In this mode, the time-out period is divided into four equal segments and the entire service sequence of the CWT must occur during the last segment (last 25% of the time-out period). If the timer is serviced anytime (any write to the CWSR register) in the first 75% of the time-out period, an immediate system reset occurs.

To prevent the core watchdog timer from interrupting or resetting, the CWSR register must be serviced by performing the following sequence:

1. Write 0x55 to CWSR.
2. Write 0xAA to CWSR.

Both writes must occur in order before the time-out, but any number of instructions can execute between the two writes. This allows interrupts and exceptions to occur, if necessary, between the two writes.

NOTE

> If the CWT is enabled and has not timed out, any write of a data value other than 0x55 or 0xAA causes an immediate system reset, regardless of the value in the CWCR[CWRI] field.

The timer value is constantly compared with the time-out period specified by CWCR[CWT], and any write to the CWCR register resets the watchdog timer. In addition, a write-once control bit in the CWCR sets the CWCR to read-only to prevent accidental updates to this control register from changing the desired system configuration. After this bit, CWCR[RO], is set, a system reset is required to clear it.

For certain values in the CWCR[CWRI] field, the CWT generates an interrupt response to a time-out. For these configurations, the SCMISR register provides a program visible interrupt request from the watchdog timer. During the interrupt service routine which handles this interrupt, the source must be explicitly cleared by writing a 0x01 to the SCMISR.

14.3.3 Core Data Fault Recovery Registers

To aid in recovery from certain types of access errors, the SCM module supports a number of registers that capture access address, attribute, and data information on bus cycles terminated with an error response. These registers can then be read during the resulting exception service routine and the appropriate recovery performed.

The details on the core fault recovery registers are provided in the above sections. It is important to note these registers are used to capture fault recovery information on any processor-initiated system bus cycle terminated with an error.

Chapter 15
Crossbar Switch (XBS)

15.1 Overview

This section provides information on the layout, configuration, and programming of the crossbar switch. The crossbar switch connects the bus masters and bus slaves using a crossbar switch structure. This structure allows bus masters to access different bus slaves simultaneously with no interference while providing arbitration among the bus masters when they access the same slave. A variety of bus arbitration methods and attributes may be programmed on a slave by slave basis.

The MCF5445*x* devices have up to seven masters and slaves (7Mx6S) connected to the crossbar switch. The seven masters are the ColdFire core, eDMA controller, FECs, USB OTG module, PCI controller, and serial boot. The slaves are SDRAM controller, FlexBus, SRAM controller ATA controller, PCI controller,backdoor, and the peripheral bus controller.

Figure 15-1 is a block diagram of the MCF5445*x* family bus architecture showing the crossbar switch configuration.

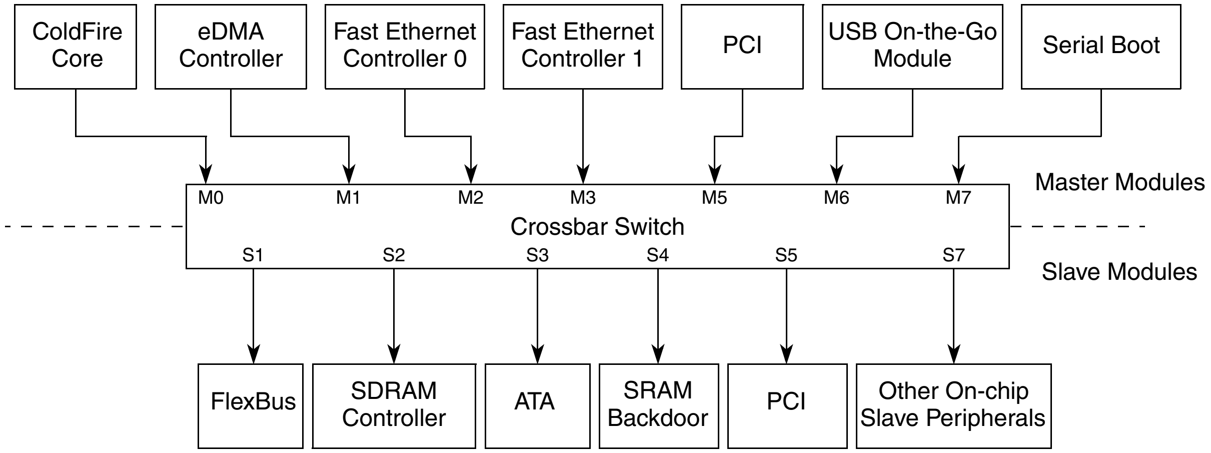


Figure 15-1. Bus Architecture Block Diagram

The modules are assigned to the numbered ports on the crossbar switch in the below table.

Table 15-1. Crossbar Switch Master/Slave Assignments

Master Modules		
Crossbar Port	**Module**	
Master 0 (M0)	ColdFire core	
Master 1 (M1)	eDMA controller	
Master 2 (M2)	Fast Ethernet controller 0	
Master 3 (M3)	Fast Ethernet controller 1	
Master 5 (M5)	PCI controller	
Master 6 (M6)	USB On-the-Go	
Master 7 (M7)	Serial boot	
Slave Modules		
Crossbar Port	**Module**	**Address Range**[1]
Slave 1 (S1)	Flexbus	0x0000_0000–0x3FFF_FFFF & 0xC000_0000–0xDFFF_FFFF
Slave 2 (S2)	SDRAM Controller	0x4000_0000–0x7FFF_FFFF
Slave 3 (S3)	ATA Controller	0x9000_0000–0x9FFF_FFFF
Slave 4 (S4)	Internal SRAM Backdoor	0x8000_0000–0x8FFF_FFFF
Slave 5 (S5)	PCI Controller	0xA000_0000–0xBFFF_FFFF
Slave 7 (S7)	Other on-chip slave peripherals	0xF000_0000–0xFFFF_FFFF[2]

[1] Unused address spaces are reserved.

[2] See the various peripheral chapters for their memory maps. Any unused space by these peripherals within this memory range is reserved and must not be accessed.

NOTE

This memory map provides two disjoint regions mapped to the FlexBus controller to support glueless connections to external memories (e.g., flash and SRAM) and a second space with one (or more) unique chip-selects that can be used for non-cacheable, non-memory devices (addresses 0xC000_0000–0xDFFF_FFFF). Additionally, this mapping is easily maps into the ColdFire access control registers, which provide a coarse association between memory addresses and their attributes (e.g., cacheable, non-cacheable). For this device, one possible configuration defines the default memory attribute as non-cacheable, and one ACR then identifies cacheable addresses, e.g., ADDR[31] equals 0 identifies the cacheable space.

15.2 Features

The crossbar switch includes these distinctive features:

- Symmetric crossbar bus switch implementation
  - — Allows concurrent accesses from different masters to different slaves
  - — Slave arbitration attributes configured on a slave by slave basis
- 32 bits wide and supports byte, word (2 byte), longword (4 byte), and 16 byte burst transfers
- Operates at a 1-to-1 clock frequency with the bus masters

15.3 Modes of Operation

The crossbar switch supports two arbitration modes (fixed or round-robin), which may be set on a slave by slave basis. Slaves configured for fixed arbitration mode have a unique arbitration level assigned to each bus master.

In fixed priority mode, the highest priority active master accessing a particular slave is granted the master bus path to that slave. A higher priority master blocks access to a given slave from a lower priority master if the higher priority master continuously requests that slave. See Section 15.5.1.1, "Fixed-Priority Operation."

In round-robin arbitration, active masters accessing a particular slave are initially granted the slave based on their master port number. Master priority is then modified in a wrap-around manner to give all masters fair access to the slave. See Section 15.5.1.2, "Round-Robin Priority Operation."

15.4 Memory Map / Register Definition

Two registers reside in each slave port of the crossbar switch. Read- and write-transfers require two bus clock cycles. The registers can only be read from and written to in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses.

A bus error response is returned if an unimplemented location is accessed within the crossbar switch. See Section 14.2.5, "SCM Interrupt Status Register (SCMISR)."

The slave registers also feature a bit that, when set, prevents the registers from being written. The registers remain readable, but future write attempts have no effect on the registers and are terminated with a bus error response to the master initiating the write. The core, for example, takes a bus error interrupt.

Table 15-2 shows the memory map for the crossbar switch program-visible registers.

Table 15-2. XBS Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4100	Priority Register Slave 1 (XBS_PRS1)	32	R/W	0x6540_3210	15.4.1/15-4
0xFC00_4110	Control Register Slave 1 (XBS_CRS1)	32	R/W	0x0000_0000	15.4.2/15-5
0xFC00_4200	Priority Register Slave 2 (XBS_PRS2)	32	R/W	0x6540_3210	15.4.1/15-4
0xFC00_4210	Control Register Slave 2 (XBS_CRS2)	32	R/W	0x0000_0000	15.4.2/15-5

Table 15-2. XBS Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4300	Priority Register Slave 3 (XBS_PRS3)	32	R/W	0x6540_3210	15.4.1/15-4
0xFC00_4310	Control Register Slave 3 (XBS_CRS3)	32	R/W	0x0000_0000	15.4.2/15-5
0xFC00_4400	Priority Register Slave 4 (XBS_PRS4)	32	R/W	0x6540_3210	15.4.1/15-4
0xFC00_4410	Control Register Slave 4 (XBS_CRS4)	32	R/W	0x0000_0000	15.4.2/15-5
0xFC00_4500	Priority Register Slave 5 (XBS_PRS5)	32	R/W	0x6540_3210	15.4.1/15-4
0xFC00_4510	Control Register Slave 5 (XBS_CRS5)	32	R/W	0x0000_0000	15.4.2/15-5
0xFC00_4700	Priority Register Slave 7 (XBS_PRS7)	32	R/W	0x6540_3210	15.4.1/15-4
0xFC00_4710	Control Register Slave 7 (XBS_CRS7)	32	R/W	0x0000_0000	15.4.2/15-5

15.4.1 XBS Priority Registers (XBS_PRS*n*)

The priority registers (XBS_PRS*n*) set the priority of each master port on a per slave port basis and reside in each slave port. The priority register can be accessed only with 32-bit accesses. After the XBS_CRS*n*[RO] bit is set, the XBS_PRS*n* register can only be read; attempts to write to it have no effect on XBS_PRS*n* and result in a bus-error response to the master initiating the write.

Additionally, no two available master ports may be programmed with the same priority level, including reserved masters. Attempts to program two or more masters with the same priority level result in a bus-error response (see Section 14.2.5, “SCM Interrupt Status Register (SCMISR)”) and the XBS_PRS*n* is not updated.

Address: 0xFC00_4100 (XBS_PRS1)
0xFC00_4200 (XBS_PRS2)
0xFC00_4300 (XBS_PRS3)
0xFC00_4400 (XBS_PRS4)
0xFC00_4500 (XBS_PRS5)
0xFC00_4700 (XBS_PRS7)

Access: Supervisor 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
R	0	M7			0	M6			0	M5			0	0	0	0	0	M3			0	M2			0	M1			0	M0		
W																																
Reset	0	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figure 15-2. XBS Priority Registers Slave *n* (XBS_PRS*n*)

Table 15-3. XBS_PRS*n* Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–28 M7	Master 7 (Serial Boot) priority. Sets the arbitration priority for this port on the associated slave port. 000 This master has level 1 (highest) priority when accessing the slave port. 001 This master has level 2 priority when accessing the slave port. 010 This master has level 3 priority when accessing the slave port. 011 This master has level 4 priority when accessing the slave port. 100 This master has level 5 priority when accessing the slave port. 101 This master has level 6 priority when accessing the slave port. 110 This master has level 7 (lowest) priority when accessing the slave port. Else Reserved
27	Reserved, must be cleared.
26–24 M6	Master 6 (USB OTG) priority. See M7 description.
23	Reserved, must be cleared.
22–20 M5	Master 5 (PCI controller) priority. See M7 description.
19–15	Reserved, must be cleared.
14–12 M3	Master 3 (FEC1) priority. See M7 description.
11	Reserved, must be cleared.
10–8 M2	Master 2 (FEC0) priority. See M7 description.
7	Reserved, must be cleared.
6–4 M1	Master 1 (eDMA) priority. See M7 description.
3	Reserved, must be cleared.
2–0 M0	Master 0 (ColdFire core) priority. See M7 description.

NOTE

The possible values for the XBS_PRS*n* fields depend on the number of masters available on the device. Because the device contains seven masters, valid values are 000 to 110. Unpredictable results occur when using the reserved setting 111.

15.4.2 XBS Control Registers (XBS_CRS*n*)

The XBS control registers (XBS_CRS*n*) control several features of each slave port and must be accessed using 32-bit accesses. After XBS_CRS*n*[RO] is set, the XBS_CRS*n* can only be read; attempts to write to it have no effect and result in an error response.

Address: 0xFC00_4110 (XBS_PRS1)
0xFC00_4210 (XBS_PRS2)
0xFC00_4310 (XBS_PRS3)
0xFC00_4410 (XBS_PRS4)
0xFC00_4510 (XBS_PRS5)
0xFC00_4710 (XBS_PRS7)

Access: Supervisor 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
R	RO[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ARB	0	0	PCTL		0	PARK		
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] After this bit is set, only a hardware reset clears it.

Figure 15-3. XBS Control Registers Slave *n* (XBS_CRS*n*)

Table 15-4. XBS_CRS*n* Field Descriptions

Field	Description
31 RO	Read only. Forces both of the slave port's registers (XBS_CRS*n* and XBS_PRS*n*) to be read-only. After set, only a hardware reset clears it. 0 Both of the slave port's registers are writeable. 1 Both of the slave port's registers are read-only and cannot be written (attempted writes have no effect on the registers and result in a bus error response).
30–9	Reserved, must be cleared.
8 ARB	Arbitration Mode. Selects the arbitration policy for the slave port. 0 Fixed priority 1 Round robin (rotating) priority
7–6	Reserved, must be cleared.
5–4 PCTL	Parking control. Determines the slave port's parking control. The low-power park feature results in an overall power savings if the slave port is not saturated; however, this forces an extra latency clock when any master tries to access the slave port while not in use because it is not parked on any master. 00 When no master makes a request, the arbiter parks the slave port on the master port defined by the PARK bit field. 01 When no master makes a request, the arbiter parks the slave port on the last master to be in control of the slave port. 10 When no master makes a request, the slave port is not parked on a master and the arbiter drives all outputs to a constant safe state. 11 Reserved.
3	Reserved, must be cleared.
2–0 PARK	Park. Determines which master port the current slave port parks on when no masters are actively making requests and the PCTL bits are cleared. 000 Park on master port M0 (ColdFire Core) 001 Park on master port M1 (eDMA Controller) 010 Park on master port M2 (FEC0) 011 Park on master port M3 (FEC1) 100 Reserved 101 Park on master port M5 (PCI Controller) 110 Park on master port M6 (USB OTG) 111 Park on master port M7 (Serial Boot)

15.5 Functional Description

15.5.1 Arbitration

The crossbar switch supports two arbitration schemes: a simple fixed-priority comparison algorithm and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

15.5.1.1 Fixed-Priority Operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the XBS_PRS*n* (priority registers). If two masters request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

When a master makes a request to a slave port, the slave port checks if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every bus transfer boundary makes certain that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the new requesting master is granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed length burst transfer or a locked transfer. In this case, the new requesting master must wait until the end of the burst transfer or locked transfer before it is granted control of the slave port.

If the new requesting master's priority level is lower than the master that currently has control of the slave port, the new requesting master is forced to wait until the current master runs one of the following cycles:

- An IDLE cycle
- A non-IDLE cycle to a location other than the current slave port.

15.5.1.2 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master port number. This priority is based on how far ahead the master port number of the requesting master is to the master port number of the current bus master for this slave. Master port numbers are compared modulo the total number of bus masters, i.e. take the requesting master port number minus the current bus master's port number modulo the total number of bus masters. The master port with the highest priority based on this comparison is granted control over the slave port at the next bus transfer boundary.

After granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line is granted access to the slave port at the next transfer boundary.

Parking may continue to be used in a round-robin mode, but does not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff occurs to the next master in line after one cycle of arbitration. If the slave port is put into low-power park mode, the round-robin pointer is reset to point at master port 0, giving it the highest priority.

15.5.1.3 Priority Assignment

Each master port needs to be assigned a unique 3-bit priority level. If an attempt is made to program multiple master ports with the same priority level within the priority registers (XBS_PRS*n*) the crossbar switch responds with a bus error (refer to Section 14.2.5, “SCM Interrupt Status Register (SCMISR)”) and the registers are not updated.

15.6 Initialization/Application Information

No initialization is required by or for the crossbar switch. Hardware reset ensures all the register bits used by the crossbar switch are properly initialized to a valid state. Settings and priorities should be programmed to achieve maximum system performance.

Chapter 16
Pin Multiplexing and Control

16.1 Introduction

Many of the pins associated with the device may be used for several different functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for their primary function, many of the pins may be used as general-purpose digital I/O (GPIO) pins. In some cases, the pin function is set by the operating mode, and the alternate pin functions are not supported.

Each GPIO port has registers that configure, monitor, and control the port pins. Figure 16-1 is a block diagram of the device ports. The GPIO functionality of the port IRQ pins is selected by the edge port module. They are shown in the figure only for completeness.

This chapter also includes registers for controlling the drive strengths and slew rates of the external pins.

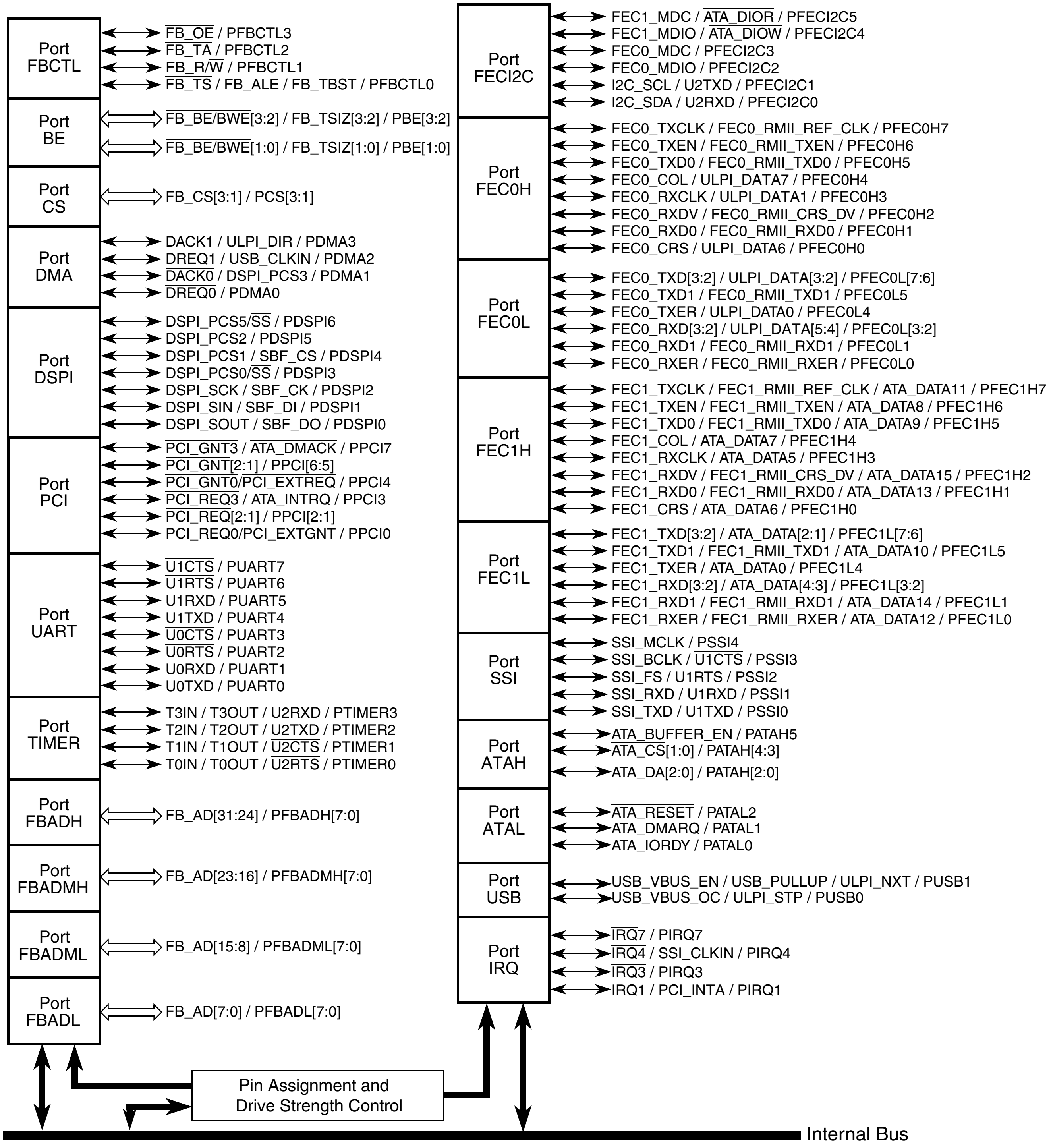

Port FBCTL
FB_OE / PFBCTL3
FB_TA / PFBCTL2
FB_R/W / PFBCTL1
FB_TS / FB_ALE / FB_TBST / PFBCTL0
Port BE
FB_BE/BWE[3:2] / FB_TSIZ[3:2] / PBE[3:2]
FB_BE/BWE[1:0] / FB_TSIZ[1:0] / PBE[1:0]
Port CS
FB_CS[3:1] / PCS[3:1]
Port DMA
DACK1 / ULPI_DIR / PDMA3
DREQ1 / USB_CLKIN / PDMA2
DACK0 / DSPI_PCS3 / PDMA1
DREQ0 / PDMA0
Port DSPI
DSPI_PCS5/SS / PDSPI6
DSPI_PCS2 / PDSPI5
DSPI_PCS1 / SBF_CS / PDSPI4
DSPI_PCS0/SS / PDSPI3
DSPI_SCK / SBF_CK / PDSPI2
DSPI_SIN / SBF_DI / PDSPI1
DSPI_SOUT / SBF_DO / PDSPI0
Port PCI
PCI_GNT3 / ATA_DMACK / PPCI7
PCI_GNT[2:1] / PPCI[6:5]
PCI_GNT0/PCI_EXTREQ / PPCI4
PCI_REQ3 / ATA_INTRQ / PPCI3
PCI_REQ[2:1] / PPCI[2:1]
PCI_REQ0/PCI_EXTGNT / PPCI0
Port UART
U1CTS / PUART7
U1RTS / PUART6
U1RXD / PUART5
U1TXD / PUART4
U0CTS / PUART3
U0RTS / PUART2
U0RXD / PUART1
U0TXD / PUART0
Port TIMER
T3IN / T3OUT / U2RXD / PTIMER3
T2IN / T2OUT / U2TXD / PTIMER2
T1IN / T1OUT / U2CTS / PTIMER1
T0IN / T0OUT / U2RTS / PTIMER0
Port FBADH
FB_AD[31:24] / PFBADH[7:0]
Port FBADMH
FB_AD[23:16] / PFBADMH[7:0]
Port FBADML
FB_AD[15:8] / PFBADML[7:0]
Port FBADL
FB_AD[7:0] / PFBADL[7:0]
Port FECI2C
FEC1_MDC / ATA_DIOR / PFECI2C5
FEC1_MDIO / ATA_DIOW / PFECI2C4
FEC0_MDC / PFECI2C3
FEC0_MDIO / PFECI2C2
I2C_SCL / U2TXD / PFECI2C1
I2C_SDA / U2RXD / PFECI2C0
Port FEC0H
FEC0_TXCLK / FEC0_RMII_REF_CLK / PFEC0H7
FEC0_TXEN / FEC0_RMII_TXEN / PFEC0H6
FEC0_TXD0 / FEC0_RMII_TXD0 / PFEC0H5
FEC0_COL / ULPI_DATA7 / PFEC0H4
FEC0_RXCLK / ULPI_DATA1 / PFEC0H3
FEC0_RXDV / FEC0_RMII_CRS_DV / PFEC0H2
FEC0_RXD0 / FEC0_RMII_RXD0 / PFEC0H1
FEC0_CRS / ULPI_DATA6 / PFEC0H0
Port FEC0L
FEC0_TXD[3:2] / ULPI_DATA[3:2] / PFEC0L[7:6]
FEC0_TXD1 / FEC0_RMII_TXD1 / PFEC0L5
FEC0_TXER / ULPI_DATA0 / PFEC0L4
FEC0_RXD[3:2] / ULPI_DATA[5:4] / PFEC0L[3:2]
FEC0_RXD1 / FEC0_RMII_RXD1 / PFEC0L1
FEC0_RXER / FEC0_RMII_RXER / PFEC0L0
Port FEC1H
FEC1_TXCLK / FEC1_RMII_REF_CLK / ATA_DATA11 / PFEC1H7
FEC1_TXEN / FEC1_RMII_TXEN / ATA_DATA8 / PFEC1H6
FEC1_TXD0 / FEC1_RMII_TXD0 / ATA_DATA9 / PFEC1H5
FEC1_COL / ATA_DATA7 / PFEC1H4
FEC1_RXCLK / ATA_DATA5 / PFEC1H3
FEC1_RXDV / FEC1_RMII_CRS_DV / ATA_DATA15 / PFEC1H2
FEC1_RXD0 / FEC1_RMII_RXD0 / ATA_DATA13 / PFEC1H1
FEC1_CRS / ATA_DATA6 / PFEC1H0
Port FEC1L
FEC1_TXD[3:2] / ATA_DATA[2:1] / PFEC1L[7:6]
FEC1_TXD1 / FEC1_RMII_TXD1 / ATA_DATA10 / PFEC1L5
FEC1_TXER / ATA_DATA0 / PFEC1L4
FEC1_RXD[3:2] / ATA_DATA[4:3] / PFEC1L[3:2]
FEC1_RXD1 / FEC1_RMII_RXD1 / ATA_DATA14 / PFEC1L1
FEC1_RXER / FEC1_RMII_RXER / ATA_DATA12 / PFEC1L0
Port SSI
SSI_MCLK / PSSI4
SSI_BCLK / U1CTS / PSSI3
SSI_FS / U1RTS / PSSI2
SSI_RXD / U1RXD / PSSI1
SSI_TXD / U1TXD / PSSI0
Port ATAH
ATA_BUFFER_EN / PATAH5
ATA_CS[1:0] / PATAH[4:3]
ATA_DA[2:0] / PATAH[2:0]
Port ATAL
ATA_RESET / PATAL2
ATA_DMARQ / PATAL1
ATA_IORDY / PATAL0
Port USB
USB_VBUS_EN / USB_PULLUP / ULPI_NXT / PUSB1
USB_VBUS_OC / ULPI_STP / PUSB0
Port IRQ
IRQ7 / PIRQ7
IRQ4 / SSI_CLKIN / PIRQ4
IRQ3 / PIRQ3
IRQ1 / PCI_INTA / PIRQ1
Pin Assignment and Drive Strength Control
Internal Bus

Figure 16-1. Ports Block Diagram

16.1.1 Overview

The external pin-muxing and control module configures various external pins, including those used for:

- External bus accesses
- External device selection
- Ethernet data and control
- I^{2}C serial control
- DSPI
- DMA
- SSI
- USB
- ATAPI
- PCI
- UART
- 32-bit DMA timers

16.1.2 Features

The module includes these distinctive features:

- Control of primary function use
  - On all supported GPIO ports, except those for FB_AD[31:0] pins
  - On pins whose GPIO is not supported by ports module: $\overline{\text{IRQ}}$[7,4,3,1]
- General purpose I/O support for all ports
  - Registers for storing output pin data
  - Registers for controlling pin data direction
  - Registers for reading current pin state
  - Registers for setting and clearing output pin data registers
- Control of functional pad drive strengths
- Slew rate control for PCI and SDRAM pins

16.2 External Signal Description

The external pins controllable by this module are listed under the GPIO column in Figure 16-2.

NOTE

In this table and throughout this document, a single signal within a group is designated without square brackets (i.e., FB_AD23), while designations for multiple signals within a group use brackets (i.e., FB_AD[23:21]) and is meant to include all signals within the two bracketed numbers when these numbers are separated by a colon.

NOTE

The primary functionality of a pin is not necessarily its default functionality. Most pins that are muxed with GPIO default to their GPIO functionality. See Table 16-1 for a list of the exceptions.

Table 16-1. Special-Case Default Signal Functionality

Pin	256 MAPBGA	360 TEPBGA
FB_AD[31:0]	FB_AD[31:0] except when serial boot selects 0-bit boot port size.	
$\overline{\text{FB\_BE/BWE}}$[3:0]	$\overline{\text{FB\_BE/BWE}}$[3:0]	
$\overline{\text{FB\_CS}}$[3:1]	$\overline{\text{FB\_CS}}$[3:1]	
$\overline{\text{FB\_OE}}$	$\overline{\text{FB\_OE}}$	
FB_R/$\overline{\text{W}}$	FB_R/$\overline{\text{W}}$	
$\overline{\text{FB\_TA}}$	$\overline{\text{FB\_TA}}$	
$\overline{\text{FB\_TS}}$	$\overline{\text{FB\_TS}}$	
$\overline{\text{PCI\_GNT}}$[3:0]	GPIO	$\overline{\text{PCI\_GNT}}$[3:0]
$\overline{\text{PCI\_REQ}}$[3:0]	GPIO	$\overline{\text{PCI\_REQ}}$[3:0]
$\overline{\text{IRQ1}}$	GPIO	$\overline{\text{PCI\_INTA}}$ and configured as an agent.
ATA_RESET	GPIO	ATA reset

Table 16-2. MCF5445*x* Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
Reset								
$\overline{\text{RESET}}$	—	—	—	U	I	EVDD	L4	Y18
$\overline{\text{RSTOUT}}$	—	—	—	—	O	EVDD	M15	B17
Clock								
EXTAL/PCI_CLK	—	—	—	—	I	EVDD	M16	A16
XTAL	—	—	—	U[3]	O	EVDD	L16	A17
Mode Selection								
BOOTMOD[1:0]	—	—	—	—	I	EVDD	M5, M7	AB17, AB21
FlexBus								

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
FB_AD[31:24]	PFBADH[7:0][4]	FB_D[31:24]	—	—	I/O	EVDD	A14, A13, D12, C12, B12, A12, D11, C11	J2, K4, J1, K1–3, L1, L4
FB_AD[23:16]	PFBADMH[7:0][4]	FB_D[23:16]	—	—	I/O	EVDD	B11, A11, D10, C10, B10, A10, D9, C9	L2, L3, M1–4, N1–2
FB_AD[15:8]	PFBADML[7:0][4]	FB_D[15:8]	—	—	I/O	EVDD	B9, A9, D8, C8, B8, A8, D7, C7	P1–2, R1–3, P4, T1–2
FB_AD[7:0]	PFBADL[7:0][4]	FB_D[7:0]	—	—	I/O	EVDD	B7, A7, D6, C6, B6, A6, D5, C5	T3–4, U1–3, V1–2, W1
FB_BE/BWE[3:2]	PBE[3:2]	FB_TSIZ[1:0]	—	—	O	EVDD	B5, A5	Y1, W2
FB_BE/BWE[1:0]	PBE[1:0]	—	—	—	O	EVDD	B4, A4	W3, Y2
FB_CLK	—	—	—	—	O	EVDD	B13	J3
FB_CS[3:1]	PCS[3:1]	—	—	—	O	EVDD	C2, D4, C3	W5, AA4, AB3
FB_CS0	—	—	—	—	O	EVDD	C4	Y4
FB_OE	PFBCTL3	—	—	—	O	EVDD	A2	AA1
FB_R/W	PFBCTL2	—	—	—	O	EVDD	B2	AA3
FB_TA	PFBCTL1	—	—	U	I	EVDD	B1	AB2
FB_TS	PFBCTL0	FB_ALE	FB_TBST	—	O	EVDD	A3	Y3
PCI Controller[5]								
PCI_AD[31:0]	—	FB_A[31:0]	—	—	I/O	EVDD	—	C11, D11, A10, B10, J4, G2, G3, F1, D12, C12, B12, A11, B11, B9, D9, D10, A8, B8, A5, B5, A4, A3, B3, D4, D3, E3–E1, F3, C2, D2, C1
—	—	FB_A[23:0]	—	—	I/O	EVDD	K14–13, J15–13, H13–15, G15–13, F14–13, E15–13, D16, B16, C15, B15, C14, D15, C16, D14	—
PCI_CBE[3:0]	—	—	—	—	I/O	EVDD	—	G4, E4, D1, B1
PCI_DEVSEL	—	—	—	—	O	EVDD	—	F2
PCI_FRAME	—	—	—	—	I/O	EVDD	—	B2
PCI_GNT3	PPCI7	ATA_DMACK	—	—	O	EVDD	—	B7
PCI_GNT[2:1]	PPCI[6:5]	—	—	—	O	EVDD	—	C8, C9

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
$\overline{\text{PCI\_GNT0}}$/ $\overline{\text{PCI\_EXTREQ}}$	PPCI4	—	—	—	O	EVDD	—	A9
PCI_IDSEL	—	—	—	—	I	EVDD	—	D5
$\overline{\text{PCI\_IRDY}}$	—	—	—	—	I/O	EVDD	—	C3
PCI_PAR	—	—	—	—	I/O	EVDD	—	C4
$\overline{\text{PCI\_PERR}}$	—	—	—	—	I/O	EVDD	—	B4
$\overline{\text{PCI\_REQ3}}$	PPCI3	ATA_INTRQ	—	—	I	EVDD	—	C7
$\overline{\text{PCI\_REQ}}$[2:1]	PPCI[2:1]	—	—	—	I	EVDD	—	D7, C5
$\overline{\text{PCI\_REQ0}}$/ $\overline{\text{PCI\_EXTGNT}}$	PPCI0	—	—	—	I	EVDD	—	A2
$\overline{\text{PCI\_RST}}$	—	—	—	—	O	EVDD	—	B6
$\overline{\text{PCI\_SERR}}$	—	—	—	—	I/O	EVDD	—	A6
$\overline{\text{PCI\_STOP}}$	—	—	—	—	I/O	EVDD	—	A7
$\overline{\text{PCI\_TRDY}}$	—	—	—	—	I/O	EVDD	—	C10
SDRAM Controller								
SD_A[13:0]	—	—	—	—	O	SDVDD	R1, P1, N2, P2, R2, T2, M4, N3, P3, R3, T3, T4, R4, N4	V22, U20–22, T19–22, R20–22, N19, P20–21
SD_BA[1:0]	—	—	—	—	O	SDVDD	P4, T5	P22, P19
$\overline{\text{SD\_CAS}}$	—	—	—	—	O	SDVDD	T6	L19
SD_CKE	—	—	—	—	O	SDVDD	N5	N22
SD_CLK	—	—	—	—	O	SDVDD	T9	L22
$\overline{\text{SD\_CLK}}$	—	—	—	—	O	SDVDD	T8	M22
$\overline{\text{SD\_CS}}$[1:0]	—	—	—	—	O	SDVDD	P6, R6	L20, M20
SD_D[31:16]	—	—	—	—	I/O	SDVDD	N6, T7, N7, P7, R7, R8, P8, N8, N9, T10, R10, P10, N10, T11, R11, P11	L21, K22, K21, K20, J20, J19, J21, J22, H20, G22, G21, G20, G19, F22, F21, F20
SD_DM[3:2]	—	—	—	—	O	SDVDD	P9, N12	H21, E21
SD_DQS[3:2]	—	—	—	—	O	SDVDD	R9, N11	H22, E22
$\overline{\text{SD\_RAS}}$	—	—	—	—	O	SDVDD	P5	N21
SD_VREF	—	—	—	—	I	SDVDD	M8	M21

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
SD_WE	—	—	—	—	O	SDVDD	R5	N20
External Interrupts Port[6]								
IRQ7	PIRQ7	—	—	—	I	EVDD	L1	ABB13
IRQ4	PIRQ4	—	SSI_CLKIN	—	I	EVDD	L2	ABB13
IRQ3	PIRQ3	—	—	—	I	EVDD	L3	AB14
IRQ1	PIRQ1	PCI_INTA	—	—	I	EVDD	F15	C6
FEC0								
FEC0_MDC	PFECI2C3	—	—	—	O	EVDD	F3	AB8
FEC0_MDIO	PFECI2C2	—	—	—	I/O	EVDD	F2	Y7
FEC0_COL	PFEC0H4	—	ULPI_DATA7	—	I	EVDD	E1	AB7
FEC0_CRS	PFEC0H0	—	ULPI_DATA6	—	I	EVDD	F1	AA7
FEC0_RXCLK	PFEC0H3	—	ULPI_DATA1	—	I	EVDD	G1	AA8
FEC0_RXDV	PFEC0H2	FEC0_RMII_ CRS_DV	—	—	I	EVDD	G2	Y8
FEC0_RXD[3:2]	PFEC0L[3:2]	—	ULPI_DATA[5:4]	—	I	EVDD	G3, G4	AB9, Y9
FEC0_RXD1	PFEC0L1	FEC0_RMII_RXD1	—	—	I	EVDD	H1	W9
FEC0_RXD0	PFEC0H1	FEC0_RMII_RXD0	—	—	I	EVDD	H2	AB10
FEC0_RXER	PFEC0L0	FEC0_RMII_RXER	—	—	I	EVDD	H3	AA10
FEC0_TXCLK	PFEC0H7	FEC0_RMII_ REF_CLK	—	—	I	EVDD	H4	Y10
FEC0_TXD[3:2]	PFEC0L[7:6]	—	ULPI_DATA[3:2]	—	O	EVDD	J1, J2	W10, AB11
FEC0_TXD1	PFEC0L5	FEC0_RMII_TXD1	—	—	O	EVDD	J3	AA11
FEC0_TXD0	PFEC0H5	FEC0_RMII_TXD0	—	—	O	EVDD	J4	Y11
FEC0_TXEN	PFEC0H6	FEC0_RMII_TXEN	—	—	O	EVDD	K1	W11
FEC0_TXER	PFEC0L4	—	ULPI_DATA0	—	O	EVDD	K2	AB12
FEC1								
FEC1_MDC	PFECI2C5	—	ATA_DIOR	—	O	EVDD	—	W20
FEC1_MDIO	PFECI2C4	—	ATA_DIOW	—	I/O	EVDD	—	Y22
FEC1_COL	PFEC1H4	—	ATA_DATA7	—	I	EVDD	—	AB18
FEC1_CRS	PFEC1H0	—	ATA_DATA6	—	I	EVDD	—	AA18

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
FEC1_RXCLK	PFEC1H3	—	ATA_DATA5	—	I	EVDD	—	W14
FEC1_RXDV	PFEC1H2	FEC1_RMII_ CRS_DV	ATA_DATA15	—	I	EVDD	—	AB15
FEC1_RXD[3:2]	PFEC1L[3:2]	—	ATA_DATA[4:3]	—	I	EVDD	—	AA15, Y15
FEC1_RXD1	PFEC1L1	FEC1_RMII_RXD1	ATA_DATA14	—	I	EVDD	—	AA17
FEC1_RXD0	PFEC1H1	FEC1_RMII_RXD0	ATA_DATA13	—	I	EVDD	—	Y17
FEC1_RXER	PFEC1L0	FEC1_RMII_RXER	ATA_DATA12	—	I	EVDD	—	W17
FEC1_TXCLK	PFEC1H7	FEC1_RMII_ REF_CLK	ATA_DATA11	—	I	EVDD	—	AB19
FEC1_TXD[3:2]	PFEC1L[7:6]	—	ATA_DATA[2:1]	—	O	EVDD	—	Y19, W18
FEC1_TXD1	PFEC1L5	FEC1_RMII_TXD1	ATA_DATA10	—	O	EVDD	—	AA19
FEC1_TXD0	PFEC1H5	FEC1_RMII_TXD0	ATA_DATA9	—	O	EVDD	—	Y20
FEC1_TXEN	PFEC1H6	FEC1_RMII_TXEN	ATA_DATA8	—	O	EVDD	—	AA21
FEC1_TXER	PFEC1L4	—	ATA_DATA0	—	O	EVDD	—	AA22
USB On-the-Go								
USB_DM	—	—	—	—	O	USB VDD	F16	A14
USB_DP	—	—	—	—	O	USB VDD	E16	A15
USB_VBUS_EN	PUSB1	USB_PULLUP	ULPI_NXT	—	O	USB VDD	E5	AA2
USB_VBUS_OC	PUSB0	—	ULPI_STP	UD[7]	I	USB VDD	B3	V4
ATA								
ATA_BUFFER_EN	PATAH5	—	—	—	O	EVDD	—	Y13
ATA_CS[1:0]	PATAH[4:3]	—	—	—	O	EVDD	—	W21, W22
ATA_DA[2:0]	PATAH[2:0]	—	—	—	O	EVDD	—	V19–21
ATA_RESET	PATAL2	—	—	—	O	EVDD	—	W13
ATA_DMARQ	PATAL1	—	—	—	I	EVDD	—	AA14
ATA_IORDY	PATAL0	—	—	—	I	EVDD	—	Y14
Real Time Clock								
EXTAL32K	—	—	—	—	I	EVDD	J16	A13
XTAL32K	—	—	—	—	O	EVDD	H16	A12

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
SSI								
SSI_MCLK	PSSI4	—	—	—	O	EVDD	T13	D20
SSI_BCLK	PSSI3	$\overline{\text{U1CTS}}$	—	—	I/O	EVDD	R13	E19
SSI_FS	PSSI2	$\overline{\text{U1RTS}}$	—	—	I/O	EVDD	P12	E20
SSI_RXD	PSSI1	U1RXD	—	UD	I	EVDD	T12	D21
SSI_TXD	PSSI0	U1TXD	—	UD	O	EVDD	R12	D22
I^2C								
I2C_SCL	PFECI2C1	—	U2TXD	U	I/O	EVDD	K3	AA12
I2C_SDA	PFECI2C0	—	U2RXD	U	I/O	EVDD	K4	Y12
DMA								
$\overline{\text{DACK1}}$	PDMA3	—	ULPI_DIR	—	O	EVDD	M14	C17
$\overline{\text{DREQ1}}$	PDMA2	—	USB_CLKIN	U	I	EVDD	P16	C18
$\overline{\text{DACK0}}$	PDMA1	DSPI_PCS3	—	—	O	EVDD	N15	A18
$\overline{\text{DREQ0}}$	PDMA0	—	—	U	I	EVDD	N16	B18
DSPI								
DSPI_PCS5/$\overline{\text{PCSS}}$	PDSPI6	—	—	—	O	EVDD	N14	D18
DSPI_PCS2	PDSPI5	—	—	—	O	EVDD	L13	A19
DSPI_PCS1	PDSPI4	$\overline{\text{SBF\_CS}}$	—	—	O	EVDD	P14	B20
DSPI_PCS0/$\overline{\text{SS}}$	PDSPI3	—	—	U	I/O	EVDD	R16	D17
DSPI_SCK	PDSPI2	SBF_CK	—	—	I/O	EVDD	R15	A20
DSPI_SIN	PDSPI1	SBF_DI	—	[8]	I	EVDD	P15	B19
DSPI_SOUT	PDSPI0	SBF_DO	—	—	O	EVDD	N13	C20
UARTs								
$\overline{\text{U1CTS}}$	PUART7	—	—	—	I	EVDD	—	V3
$\overline{\text{U1RTS}}$	PUART6	—	—	—	O	EVDD	—	U4
U1RXD	PUART5	—	—	—	I	EVDD	—	P3
U1TXD	PUART4	—	—	—	O	EVDD	—	N3
$\overline{\text{U0CTS}}$	PUART3	—	—	—	I	EVDD	M3	Y16
$\overline{\text{U0RTS}}$	PUART2	—	—	—	O	EVDD	M2	AA16

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
U0RXD	PUART1	—	—	—	I	EVDD	N1	AB16
U0TXD	PUART0	—	—	—	O	EVDD	M1	W15
Note: The UART1 and UART 2 signals are multiplexed on the DMA timers and I2C pins.								
DMA Timers								
DT3IN	PTIMER3	DT3OUT	U2RXD	—	I	EVDD	C13	H2
DT2IN	PTIMER2	DT2OUT	U2TXD	—	I	EVDD	D13	H1
DT1IN	PTIMER1	DT1OUT	$\overline{\text{U2CTS}}$	—	I	EVDD	B14	H3
DT0IN	PTIMER0	DT0OUT	$\overline{\text{U2RTS}}$	—	I	EVDD	A15	G1
BDM/JTAG[9]								
PSTDDATA[7:0]	—	—	—	—	O	EVDD	E2, D1, F4, E3, D2, C1, E4, D3	AA6, AB6, AB5, W6, Y6, AA5, AB4, Y5
JTAG_EN	—	—	—	D	I	EVDD	M11	C21
PSTCLK	—	TCLK	—	—	I	EVDD	P13	C22
DSI	—	TDI	—	U	I	EVDD	T15	C19
DSO	—	TDO	—	—	O	EVDD	T14	A21
$\overline{\text{BKPT}}$	—	TMS	—	U	I	EVDD	R14	B21
DSCLK	—	$\overline{\text{TRST}}$	—	U	I	EVDD	M13	B22
Test								
TEST	—	—	—	D	I	EVDD	M6	AB20
PLLTEST	—	—	—	—	O	EVDD	K16	D15
Power Supplies								
IVDD	—	—	—	—	—	—	E6–12, F5, F12	D6, D8, D14, F4, H4, N4, R4, W4, W7, W8, W12, W16, W19
EVDD	—	—	—	—	—	—	G5, G12, H5, H12, J5, J12, K5, K12, L5–6, L12	D13, D19, G8, G11, G14, G16, J7, J16, L7, L16, N16, P7, R16, T8, T12, T14, T16
SD_VDD	—	—	—	—	—	—	L7–11, M9, M10	F19, H19, K19, M19, R19, U19
VDD_OSC	—	—	—	—	—	—	L14	B16
VDD_A_PLL	—	—	—	—	—	—	K15	C14

Table 16-2. MCF5445*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF54450 MCF54451 256 MAPBGA	MCF54452 MCF54453 MCF54454 MCF54455 360 TEPBGA
VDD_RTC	—	—	—	—	—	—	M12	C13
VSS	—	—	—	—	—	—	A1, A16, F6–11, G6–11, H6–11, J6–11, K6–11, T1, T16	A1, A22, B14, G7, G9–10, G12–13, G15, H7, H16, J9–14, K7, K9–14, K16, L9–14, M7, M9–M14, M16, N9–14, P9–14, P16, R7, T7, T9–11, T13, T15, AB1, AB22
VSS_OSC	—	—	—	—	—	—	L15	C16

[1] Pull-ups are generally only enabled on pins with their primary function, except as noted.

[2] Refers to pin’s primary function.

[3] Enabled only in oscillator bypass mode (internal crystal oscillator is disabled).

[4] Serial boot must select 0-bit boot port size to enable the GPIO mode on these pins.

[5] When the PCI is enabled, all PCI bus pins come up configured as such. This includes the PCI_GNT and PCI_REQ lines, which have GPIO. The IRQ1/$\overline{\text{PCI\_INTA}}$ signal is a special case. It comes up as $\overline{\text{PCI\_INTA}}$ when booting as a PCI agent and as GPIO when booting as a PCI host.
For the 360 TEPBGA, booting with PCI disabled results in all dedicated PCI pins being safe-stated. The $\overline{\text{PCI\_GNT}}$ and $\overline{\text{PCI\_REQ}}$ lines and IRQ1/$\overline{\text{PCI\_INTA}}$ come up as GPIO.

[6] GPIO functionality is determined by the edge port module. The pin multiplexing and control module is only responsible for assigning the alternate functions.

[7] Depends on programmed polarity of the USB_VBUS_OC signal.

[8] Pull-up when the serial boot facility (SBF) controls the pin

[9] If JTAG_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The pin multiplexing and control module is not responsible for assigning these pins.

Refer to the Chapter 2, “Signal Descriptions,” for more detailed descriptions of these pins and other pins not controlled by this module. The function of most of the pins (primary function, GPIO, etc.) is determined by the pin assignment registers (PAR_*x*).

From the above table, there are several cases where a function is available on more than one pin. While it is possible to enable the function on more than one pin simultaneously, this type of programming should be avoided for input functions to prevent unexpected behavior. All multiple-pin functions are listed in Table 16-3.

Table 16-3. Multiple-Pin Functions

Function	Direction	Associated Pins
$\overline{\text{U1CTS}}$	I	$\overline{\text{U1CTS}}$, SSI_BCLK
$\overline{\text{U1RTS}}$	O	$\overline{\text{U1RTS}}$, SSI_FS
U1RXD	I	U1RXD, SSI_RXD
U1TXD	O	U1TXD, SSI_TXD

Table 16-3. Multiple-Pin Functions (continued)

Function	Direction	Associated Pins
U2RXD	I	I2C_SDA, T3IN
U2TXD	O	I2C_SCL, T2IN

16.3 Memory Map/Register Definition

Table 16-4 summarizes all the registers in the pin multiplexing and control address space.

Table 16-4. Pin Multiplexing and Control Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Port Output Data Registers					
0xFC0A_4000	PODR_FEC0H	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4001	PODR_FEC0L	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4002	PODR_SSI	8	R/W	0x1F	16.3.1/16-16
0xFC0A_4003	PODR_FBCTL	8	R/W	0x0F	16.3.1/16-16
0xFC0A_4004	PODR_BE	8	R/W	0x0F	16.3.1/16-16
0xFC0A_4005	PODR_CS	8	R/W	0x0E	16.3.1/16-16
0xFC0A_4006	PODR_DMA	8	R/W	0x0F	16.3.1/16-16
0xFC0A_4007	PODR_FECI2C	8	R/W	0x3F	16.3.1/16-16
0xFC0A_4009	PODR_UART	8	R/W	0xFF	16.3.1/16-16
0xFC0A_400A	PODR_DSPI	8	R/W	0x7F	16.3.1/16-16
0xFC0A_400B	PODR_TIMER	8	R/W	0x0F	16.3.1/16-16
0xFC0A_400C	PODR_PCI	8	R/W	0xFF	16.3.1/16-16
0xFC0A_400D	PODR_USB	8	R/W	0x03	16.3.1/16-16
0xFC0A_400E	PODR_ATAH	8	R/W	0x3F	16.3.1/16-16
0xFC0A_400F	PODR_ATAL	8	R/W	0x07	16.3.1/16-16
0xFC0A_4010	PODR_FEC1H	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4011	PODR_FEC1L	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4014	PODR_FBADH	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4015	PODR_FBADMH	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4016	PODR_FBADML	8	R/W	0xFF	16.3.1/16-16
0xFC0A_4017	PODR_FBADL	8	R/W	0xFF	16.3.1/16-16
Port Data Direction Registers					
0xFC0A_4018	PDDR_FEC0H	8	R/W	0x00	16.3.2/16-18

Table 16-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4019	PDDR_FEC0L	8	R/W	0x00	16.3.2/16-18
0xFC0A_401A	PDDR_SSI	8	R/W	0x00	16.3.2/16-18
0xFC0A_401B	PDDR_FBCTL	8	R/W	0x00	16.3.2/16-18
0xFC0A_401C	PDDR_BE	8	R/W	0x00	16.3.2/16-18
0xFC0A_401D	PDDR_CS	8	R/W	0x00	16.3.2/16-18
0xFC0A_401E	PDDR_DMA	8	R/W	0x00	16.3.2/16-18
0xFC0A_401F	PDDR_FECI2C	8	R/W	0x00	16.3.2/16-18
0xFC0A_4021	PDDR_UART	8	R/W	0x00	16.3.2/16-18
0xFC0A_4022	PDDR_DSPI	8	R/W	0x00	16.3.2/16-18
0xFC0A_4023	PDDR_TIMER	8	R/W	0x00	16.3.2/16-18
0xFC0A_4024	PDDR_PCI	8	R/W	0x00	16.3.2/16-18
0xFC0A_4025	PDDR_USB	8	R/W	0x00	16.3.2/16-18
0xFC0A_4026	PDDR_ATAH	8	R/W	0x00	16.3.2/16-18
0xFC0A_4027	PDDR_ATAL	8	R/W	0x00	16.3.2/16-18
0xFC0A_4028	PDDR_FEC1H	8	R/W	0x00	16.3.2/16-18
0xFC0A_4029	PDDR_FEC1L	8	R/W	0x00	16.3.2/16-18
0xFC0A_402C	PDDR_FBADH	8	R/W	0x00	16.3.2/16-18
0xFC0A_402D	PDDR_FBADMH	8	R/W	0x00	16.3.2/16-18
0xFC0A_402E	PDDR_FBADML	8	R/W	0x00	16.3.2/16-18
0xFC0A_402F	PDDR_FBADL	8	R/W	0x00	16.3.2/16-18
Port Pin Data/Set Data Registers					
0xFC0A_4030	PPDSDR_FEC0H	8	R/W	See Section	16.3.3/16-20
0xFC0A_4031	PPDSDR_FEC0L	8	R/W	See Section	16.3.3/16-20
0xFC0A_4032	PPDSDR_SSI	8	R/W	See Section	16.3.3/16-20
0xFC0A_4033	PPDSDR_FBCTL	8	R/W	See Section	16.3.3/16-20
0xFC0A_4034	PPDSDR_BE	8	R/W	See Section	16.3.3/16-20
0xFC0A_4035	PPDSDR_CS	8	R/W	See Section	16.3.3/16-20
0xFC0A_4036	PPDSDR_DMA	8	R/W	See Section	16.3.3/16-20
0xFC0A_4037	PPDSDR_FECI2C	8	R/W	See Section	16.3.3/16-20
0xFC0A_4039	PPDSDR_UART	8	R/W	See Section	16.3.3/16-20
0xFC0A_403A	PPDSDR_DSPI	8	R/W	See Section	16.3.3/16-20
0xFC0A_403B	PPDSDR_TIMER	8	R/W	See Section	16.3.3/16-20

Table 16-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_403C	PPDSDR_PCI	8	R/W	See Section	16.3.3/16-20
0xFC0A_403D	PPDSDR_USB	8	R/W	See Section	16.3.3/16-20
0xFC0A_403E	PPDSDR_ATAH	8	R/W	See Section	16.3.3/16-20
0xFC0A_403F	PPDSDR_ATAL	8	R/W	See Section	16.3.3/16-20
0xFC0A_4040	PPDSDR_FEC1H	8	R/W	See Section	16.3.3/16-20
0xFC0A_4041	PPDSDR_FEC1L	8	R/W	See Section	16.3.3/16-20
0xFC0A_4044	PPDSDR_FBADH	8	R/W	See Section	16.3.3/16-20
0xFC0A_4045	PPDSDR_FBADMH	8	R/W	See Section	16.3.3/16-20
0xFC0A_4046	PPDSDR_FBADML	8	R/W	See Section	16.3.3/16-20
0xFC0A_4047	PPDSDR_FBADL	8	R/W	See Section	16.3.3/16-20
Port Clear Output Data Registers					
0xFC0A_4048	PCLRR_FEC0H	8	W	0x00	16.3.4/16-23
0xFC0A_4049	PCLRR_FEC0L	8	W	0x00	16.3.4/16-23
0xFC0A_404A	PCLRR_SSI	8	W	0x00	16.3.4/16-23
0xFC0A_404B	PCLRR_FBCTL	8	W	0x00	16.3.4/16-23
0xFC0A_404C	PCLRR_BE	8	W	0x00	16.3.4/16-23
0xFC0A_404D	PCLRR_CS	8	W	0x00	16.3.4/16-23
0xFC0A_404E	PCLRR_DMA	8	W	0x00	16.3.4/16-23
0xFC0A_404F	PCLRR_FECI2C	8	W	0x00	16.3.4/16-23
0xFC0A_4051	PCLRR_UART	8	W	0x00	16.3.4/16-23
0xFC0A_4052	PCLRR_DSPI	8	W	0x00	16.3.4/16-23
0xFC0A_4053	PCLRR_TIMER	8	W	0x00	16.3.4/16-23
0xFC0A_4054	PCLRR_PCI	8	W	0x00	16.3.4/16-23
0xFC0A_4055	PCLRR_USB	8	W	0x00	16.3.4/16-23
0xFC0A_4056	PCLRR_ATAH	8	W	0x00	16.3.4/16-23
0xFC0A_4057	PCLRR_ATAL	8	W	0x00	16.3.4/16-23
0xFC0A_4058	PCLRR_FEC1H	8	W	0x00	16.3.4/16-23
0xFC0A_405A	PCLRR_FEC1L	8	W	0x00	16.3.4/16-23
0xFC0A_405C	PCLRR_FBADH	8	W	0x00	16.3.4/16-23
0xFC0A_405D	PCLRR_FBADMH	8	W	0x00	16.3.4/16-23
0xFC0A_405E	PCLRR_FBADML	8	W	0x00	16.3.4/16-23
0xFC0A_405F	PCLRR_FBADL	8	W	0x00	16.3.4/16-23

Table 16-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Pin Assignment Registers					
0xFC0A_4060	PAR_FEC	8	R/W	0x00	16.3.5.1/16-25
0xFC0A_4061	PAR_DMA	8	R/W	0x00	16.3.5.2/16-27
0xFC0A_4062	PAR_FBCTL	8	R/W	0xF8	16.3.5.3/16-27
0xFC0A_4063	PAR_DSPI	8	R/W	0x00	16.3.5.4/16-28
0xFC0A_4064	PAR_BE	8	R/W	0xF5	16.3.5.5/16-29
0xFC0A_4065	PAR_CS	8	R/W	0x0E	16.3.5.6/16-30
0xFC0A_4066	PAR_TIMER	8	R/W	0x00	16.3.5.7/16-30
0xFC0A_4067	PAR_USB	8	R/W	0x00	16.3.5.8/16-31
0xFC0A_4069	PAR_UART	8	R/W	0x00	16.3.5.9/16-32
0xFC0A_406A	PAR_FECI2C	16	R/W	0x0000	16.3.5.10/16-33
0xFC0A_406C	PAR_SSI	16	R/W	0x0000	16.3.5.11/16-34
0xFC0A_406E	PAR_ATA	16	R/W	0x0004	16.3.5.12/16-34
0xFC0A_4070	PAR_IRQ	8	R/W	See Section	16.3.5.13/16-36
0xFC0A_4072	PAR_PCI	16	R/W	See Section	16.3.5.14/16-36
Mode Select Control Registers					
0xFC0A_4074	MSCR_SDRAM	8	R/W	0xFF	16.3.6/16-38
0xFC0A_4075	MSCR_PCI	8	R/W	See Section	16.3.7/16-39
Drive Strength Control Registers					
0xFC0A_4078	DSCR_I2C	8	R/W	0x03	16.3.8/16-39
0xFC0A_4079	DSCR_FLEXBUS	8	R/W	0xFF	16.3.8/16-39
0xFC0A_407A	DSCR_FEC	8	R/W	0x0F	16.3.8/16-39
0xFC0A_407B	DSCR_UART	8	R/W	0x0F	16.3.8/16-39
0xFC0A_407C	DSCR_DSPI	8	R/W	0x03	16.3.8/16-39
0xFC0A_407D	DSCR_TIMER	8	R/W	0x03	16.3.8/16-39
0xFC0A_407E	DSCR_SSI	8	R/W	0x03	16.3.8/16-39
0xFC0A_407F	DSCR_DMA	8	R/W	0x03	16.3.8/16-39
0xFC0A_4080	DSCR_DEBUG	8	R/W	0x03	16.3.8/16-39
0xFC0A_4081	DSCR_RESET	8	R/W	0x03	16.3.8/16-39
0xFC0A_4082	DSCR_IRQ	8	R/W	0x03	16.3.8/16-39

Table 16-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4083	DSCR_USB	8	R/W	0x03	16.3.8/16-39
0xFC0A_4084	DSCR_ATA	8	R/W	0x03	16.3.8/16-39

16.3.1 Port Output Data Registers (PODR_*x*)

The PODR_*x* registers store the data to be driven on the corresponding port pins when the pins are configured for general purpose output. The PODR_*x* registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the below figures. The PODR_*x* registers are read/write. At reset, all implemented bits in the PODR_*x* registers are set. Reserved bits always remain cleared.

Reading a PODR_*x* register returns the current values in the register, not the port pin values. To set bits in a PODR_*x* register, set the PODR_*x* bits or set the corresponding bits in the PPDSDR_*x* register. To clear bits in a PODR_*x* register, clear the PODR_*x* bits or clear the corresponding bits in the PCLRR_*x* register.

Address: 0xFC0A_4000 (PODR_FEC0H)
0xFC0A_4001 (PODR_FEC0L)
0xFC0A_4009 (PODR_UART)
0xFC0A_400C (PODR_PCI)
0xFC0A_4010 (PODR_FEC1H)
0xFC0A_4011 (PODR_FEC1L)
0xFC0A_4014 (PODR_FBADH)
0xFC0A_4015 (PODR_FBADMH)
0xFC0A_4016 (PODR_FBADML)
0xFC0A_4017 (PODR_FBADL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	PODR_*x*							
W								
Reset:	1	1	1	1	1	1	1	1

Figure 16-2. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_400A (PODR_DSPI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PODR_DSPI						
W								
Reset:	0	1	1	1	1	1	1	1

Figure 16-3. Port DSPI Output Data Registers (PODR_DSPI)

Address: 0xFC0A_4007 (PODR_FECI2C)
0xFC0A_400E (PODR_ATAH)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PODR_*x*					
W								
Reset:	0	0	1	1	1	1	1	1

Figure 16-4. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4002 (PODR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PODR_SSI				
W								
Reset:	0	0	0	1	1	1	1	1

Figure 16-5. Port SSI Output Data Registers (PODR_SSI)

Address: 0xFC0A_4003 (PODR_FBCTL)
0xFC0A_4004 (PODR_BE)
0xFC0A_4006 (PODR_DMA)
0xFC0A_400B (PODR_TIMER)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PODR_*x*			
W								
Reset:	0	0	0	0	1	1	1	1

Figure 16-6. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_400F (PODR_ATAL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	PODR_ATAL		
W								
Reset:	0	0	0	0	0	1	1	1

Figure 16-7. Port ATAL Output Data Registers (PODR_ATAL)

Address: 0xFC0A_4005 (PODR_CS)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PODR_CS			0
W								
Reset:	0	0	0	0	1	1	1	0

Figure 16-8. Port CS Output Data Registers (PODR_CS)

Address: 0xFC0A_400D (PODR_USB)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PODR_USB	
W								
Reset:	0	0	0	0	0	0	1	1

Figure 16-9. Port USB Output Data Registers (PODR_USB)

Table 16-5. PODR_*x* Field Descriptions

Field	Description
PODR_*x*	Port *x* output data bits. 0 Drives 0 when the port *x* pin is general purpose output 1 Drives 1 when the port *x* pin is general purpose output

Note: See above figures for bit field positions.

16.3.2 Port Data Direction Registers (PDDR_*x*)

The PDDRs control the direction of the port pin drivers when the pins are configured for GPIO. The PDDR_*x* registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the figures below.

The PDDRs are read/write. At reset, all bits in the PDDRs are cleared. Setting any bit in a PDDR_*x* register configures the corresponding port pin as an output. Clearing any bit in a PDDR_*x* register configures the corresponding pin as an input.

Address: 0xFC0A_4018 (PDDR_FEC0H)
0xFC0A_4019 (PDDR_FEC0L)
0xFC0A_4021 (PDDR_UART)
0xFC0A_4024 (PDDR_PCI)
0xFC0A_4028 (PDDR_FEC1H)
0xFC0A_4029 (PDDR_FEC1L)
0xFC0A_402C (PDDR_FBADH)
0xFC0A_402D (PDDR_FBADMH)
0xFC0A_402E (PDDR_FBADML)
0xFC0A_402F (PDDR_FBADL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	PDDR_*x*							
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-10. Port Data Direction Registers (PDDR_*x*)

Address: 0xFC0A_4022 (PDDR_DSPI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PDDR_DSPI						
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-11. Port DSPI Data Direction Registers (PDDR_DSPI)

Address: 0xFC0A_401F (PDDR_FECI2C)
0xFC0A_4026 (PDDR_ATAH)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PDDR_*x*					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-12. Port Data Direction Registers (PDDR_*x*)

Address: 0xFC0A_401A (PDDR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PDDR_SSI				
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-13. Port PWM Output Data Registers (PDDR_SSI)

Address: 0xFC0A_401B (PDDR_FBCTL)
0xFC0A_401C (PDDR_BE)
0xFC0A_401E (PDDR_DMA)
0xFC0A_4023 (PDDR_TIMER)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PDDR_*x*			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-14. Port *x* Output Data Registers (PDDR_*x*)

Address: 0xFC0A_4027 (PDDR_ATAL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	PDDR_ATAL		
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-15. Port ATAL Output Data Registers (PDDR_ATAL)

Address: 0xFC0A_401D (PDDR_CS) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PDDR_CS			0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-16. Port CS Output Data Registers (PDDR_CS)

Address: 0xFC0A_4025 (PDDR_USB) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PDDR_USB	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-17. Port USB Output Data Registers (PDDR_USB)

Table 16-6. PDDR_*x* Field Descriptions

Field	Description
PDDR_*x*	Port *x* output data direction bits. 1 Port *x* pin configured as output 0 Port *x* pin configured as input

Note: See above figures for bit field positions.

16.3.3 Port Pin Data/Set Data Registers (PPDSDR_*x*)

The PPDSDR registers reflect the current pin states and control the setting of output pins when the pin is configured for GPIO. The PPDSDR_*x* registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the below figures.

The PPDSDR_*x* registers are read/write. At reset, the bits in the PPDSDR_*x* registers are set to the current pin states. Reading a PPDSDR_*x* register returns the current state of the port *x* pins. Setting a PPDSDR_*x* register sets the corresponding bits in the PODR_*x* register. Writing zeroes has no effect.

Address: 0xFC0A_4030 (PPDSDR_FEC0H)
0xFC0A_4031 (PPDSDR_FEC0L)
0xFC0A_4039 (PPDSDR_UART)
0xFC0A_403C (PPDSDR_PCI)
0xFC0A_4040 (PPDSDR_FEC1H)
0xFC0A_4041 (PPDSDR_FEC1L)
0xFC0A_4044 (PPDSDR_FBADH)
0xFC0A_4045 (PPDSDR_FBADMH)
0xFC0A_4046 (PPDSDR_FBADML)
0xFC0A_4047 (PPDSDR_FBADL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	PPDR_*x*							
W	PSDR_*x*							
Reset:	[P*x*7]	[P*x*6]	[P*x*5]	[P*x*4]	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 16-18. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_403A (PPDSDR_DSPI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PPDR_DSPI						
W	▨	PSDR_DSPI						
Reset:	0	[PDSPI6]	[PDSPI5]	[PDSPI4]	[PDSPI3]	[PDSPI2]	[PDSPI1]	[PDSPI0]

Figure 16-19. Port DSPI Pin Data/Set Data Registers (PPDSDR_DSPI)

Address: 0xFC0A_4037 (PPDSDR_FECI2C)
0xFC0A_403E (PPDSDR_ATAH)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PPDR_*x*					
W	▨	▨	PSDR_*x*					
Reset:	0	0	[P*x*5]	[P*x*4]	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 16-20. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_4032 (PPDSDR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PPDR_SSI				
W	▨	▨	▨	PSDR_SSI				
Reset:	0	0	0	[PSSI4]	[PSSI3]	[PSSI2]	[PSSI1]	[PSSI0]

Figure 16-21. Port SSI Pin Data/Set Data Registers (PPDSDR_SSI)

Address: 0xFC0A_4033 (PPDSDR_FBCTL)
0xFC0A_4034 (PPDSDR_BE)
0xFC0A_4036 (PPDSDR_DMA)
0xFC0A_403B (PPDSDR_TIMER)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PPDR_*x*			
W					PSDR_*x*			
Reset:	0	0	0	0	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 16-22. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*

Address: 0xFC0A_403F (PPDSDR_ATAL)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	PPDR_ATAL		
W						PSDR_ATAL		
Reset:	0	0	0	0	0	[PATAL2]	[PATAL1]	[PATAL0]

Figure 16-23. Port ATAL Pin Data/Set Data Registers (PPDSDR_ATAL)

Address: 0xFC0A_4035 (PPDSDR_CS)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PPDR_CS			0
W					PSDR_CS			
Reset:	0	0	0	0	[PCS3]	[PCS2]	[PCS1]	0

Figure 16-24. Port CS Pin Data/Set Data Registers (PPDSDR_CS)

Address: 0xFC0A_403D (PPDSDR_USB)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PPDR_USB	
W							PSDR_USB	
Reset:	0	0	0	0	0	0	[PUSB1]	[PUSB0]

Figure 16-25. Port USB Pin Data/Set Data Registers (PPDSDR_USB)

Table 16-7. PPDSDR_*x* Field Descriptions

Field	Description
PPDR_*x* (read)	Port *x* pin data bits. 0 Port *x* pin state is 0 1 Port *x* pin state is 1

Table 16-7. PPDSDR_*x* Field Descriptions (continued)

Field	Description
PSDR_*x* (write)	Port *x* set data bits. 0 No effect. 1 Set corresponding PODR_*x* bit.

Note: See above figures for bit field positions.

16.3.4 Port Clear Output Data Registers (PCLRR_*x*)

Clearing a PCLRR_*x* register clears the corresponding bits in the PODR_*x* register. Setting it has no effect. Reading the PCLRR_*x* register returns zeroes. The PCLRR_*x* registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the figures below.

Address: 0xFC0A_4048 (PCLRR_FEC0H)
0xFC0A_4049 (PCLRR_FEC0L)
0xFC0A_4051 (PCLRR_UART)
0xFC0A_4054 (PCLRR_PCI)
0xFC0A_4058 (PCLRR_FEC1H)
0xFC0A_4059 (PCLRR_FEC1L)
0xFC0A_405C (PCLRR_FBADH)
0xFC0A_405D (PCLRR_FBADMH)
0xFC0A_405E (PCLRR_FBADML)
0xFC0A_405F (PCLRR_FBADL)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	PCLRR_*x*							
Reset:	0	0	0	0	0	0	0	0

Figure 16-26. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_4052 (PCLRR_DSPI)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	▨	PCLRR_DSPI						
Reset:	0	0	0	0	0	0	0	0

Figure 16-27. Port DSPI Clear Output Data Registers (PCLRR_DSPI)

Address: 0xFC0A_404F (PCLRR_FECI2C)
0xFC0A_4056 (PCLRR_ATAH)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W			PCLRR_*x*					
Reset:	0	0	0	0	0	0	0	0

Figure 16-28. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_404A (PCLRR_SSI)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W				PCLRR_SSI				
Reset:	0	0	0	0	0	0	0	0

Figure 16-29. Port SSI Clear Output Data Registers (PCLRR_SSI)

Address: 0xFC0A_4046 (PCLRR_FBCTL)
0xFC0A_404C (PCLRR_BE)
0xFC0A_404E (PCLRR_DMA)
0xFC0A_4053 (PCLRR_TIMER)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W					PCLRR_*x*			
Reset:	0	0	0	0	0	0	0	0

Figure 16-30. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_4057 (PCLRR_ATAL)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0		0	0	0	0
W						PCLRR_ATAL		
Reset:	0	0	0	0	0	0	0	0

Figure 16-31. Port ATAL Clear Output Data Registers (PCLRR_ATAL)

Address: 0xFC0A_404D (PCLRR_CS)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W					PCLRR_CS			
Reset:	0	0	0	0	0	0	0	0

Figure 16-32. Port CS Clear Output Data Registers (PCLRR_CS)

Address: 0xFC0A_4055 (PCLRR_USB) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W							PCLRR_USB	
Reset:	0	0	0	0	0	0	0	0

Figure 16-33. Port USB Clear Output Data Registers (PCLRR_USB)

Table 16-8. PCLRR_*x* Field Descriptions

Field	Description
PCLRR_*x*	Port *x* clear data bits. 0 Clears corresponding PODR_*x* bit 1 No effect

Note: See above figures for bit field positions.

16.3.5 Pin Assignment Registers (PAR_*x*)

The pin assignment registers control which functions are currently active on the external pins. All pin assignment registers are read/write.

16.3.5.1 FEC Pin Assignment Register (PAR_FEC)

The PAR_FEC register controls the functions of the FEC0 and FEC1 pins.

Address: 0xFC0A_4060 (PAR_FEC) Access: User read/write

	7	6	5	4	3	2	1	0
R	0		PAR_FEC1		0		PAR_FEC0	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-34. FEC Pin Assignment (PAR_FEC)

Table 16-9. PAR_FEC Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, should be cleared.</td></tr>
<tr><td>6–4
PAR_FEC1</td><td>FEC1 pin assignment. Configures the FEC1_COL, FEC1_RXCLK, FEC1_RXDV, FEC1_RXD[3:0], FEC1_TXCLK, FEC1_TXD[3:0], FEC1_TXEN, FEC1_CRS, FEC1_RXER, and FEC1_TXER pins for one of their primary functions or GPIO.
<table>
<tr><th>PAR_FEC1</th><th>Meaning</th></tr>
<tr><td>000</td><td>FEC1 pins configured for GPIO</td></tr>
<tr><td>001</td><td>FEC1 pins configured for ATA data functions</td></tr>
<tr><td>010</td><td>FEC1 pins configured for FEC RMII functions;
MII-only pins are configured for ATA data functions</td></tr>
<tr><td>011</td><td>FEC1 pins configured for FEC RMII functions;
MII-only pins are configured for GPIO</td></tr>
<tr><td>100</td><td>Reserved</td></tr>
<tr><td>101</td><td>Reserved</td></tr>
<tr><td>110</td><td>Reserved</td></tr>
<tr><td>111</td><td>FEC1 pins configured for FEC MII functions</td></tr>
</table></td></tr>
<tr><td>3</td><td>Reserved, should be cleared.</td></tr>
<tr><td>2–0
PAR_FEC0</td><td>FEC0 pin assignment. Configures the FEC0_COL, FEC0_RXCLK, FEC0_RXDV, FEC0_RXD[3:0], FEC0_TXCLK, FEC0_TXD[3:0], FEC0_TXEN, FEC0_CRS, FEC0_RXER, and FEC0_TXER pins for one of their primary functions or GPIO.
<table>
<tr><th>PAR_FEC0</th><th>Meaning</th></tr>
<tr><td>000</td><td>FEC0 pins configured for GPIO</td></tr>
<tr><td>001</td><td>FEC0 pins configured for ULPI data functions;
MII pins are configured for GPIO</td></tr>
<tr><td>010</td><td>FEC0 pins configured for FEC RMII functions;
MII-only pins are configured for ULPI data functions</td></tr>
<tr><td>011</td><td>FEC0 pins configured for FEC RMII functions;
MII-only pins are configured for GPIO</td></tr>
<tr><td>100</td><td>Reserved</td></tr>
<tr><td>101</td><td>Reserved</td></tr>
<tr><td>110</td><td>Reserved</td></tr>
<tr><td>111</td><td>FEC0 pins configured for FEC MII functions</td></tr>
</table></td></tr>
</table>

16.3.5.2 DMA Pin Assignment Register (PAR_DMA)

The PAR_DMA register controls the functions of the DMA pins.

Address: 0xFC0A_4061 (PAR_DMA) Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_DACK1		PAR_DREQ1		PAR_DACK0		0	PAR_ DREQ0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-35. DMA Pin Assignment (PAR_DMA)

Table 16-10. PAR_DMA Field Descriptions

Field	Description
7–6 PAR_DACK1	$\overline{\text{DACK1}}$ pin assignment. Configures the $\overline{\text{DACK1}}$ pin for one of its primary functions or GPIO. 00 $\overline{\text{DACK1}}$ configured as GPIO 01 $\overline{\text{DACK1}}$ configured as ULPI data direction function 10 Reserved 11 $\overline{\text{DACK1}}$ configured for DMA acknowledge 1 function
5–4 PAR_DREQ1	$\overline{\text{DREQ1}}$ pin assignment. Configures the $\overline{\text{DREQ1}}$ pin for one of its primary functions or GPIO. 00 $\overline{\text{DREQ1}}$ configured as GPIO 01 $\overline{\text{DREQ1}}$ configured as USB input clock function 10 Reserved 11 $\overline{\text{DREQ1}}$ configured for DMA request 1 function
3–2 PAR_DACK0	$\overline{\text{DACK0}}$ pin assignment. Configures the $\overline{\text{DACK0}}$ pin for one of its primary functions or GPIO. 00 $\overline{\text{DACK0}}$ configured as GPIO 01 Reserved 10 $\overline{\text{DACK0}}$ configured as DSPI peripheral chip select 3 function 11 $\overline{\text{DACK0}}$ configured for DMA acknowledge 0 function
1	Reserved, should be cleared.
2–0 PAR_DREQ0	$\overline{\text{DREQ0}}$ pin assignment. Configures the $\overline{\text{DREQ0}}$ pin for its primary function or GPIO. 0 $\overline{\text{DREQ0}}$ configured as GPIO 1 $\overline{\text{DREQ0}}$ configured for DMA request 0 function

16.3.5.3 FlexBus Control Pin Assignment Register (PAR_FBCTL)

The PAR_FBCTL register controls the functions of the external FlexBus control signal pins. After reset, the FlexBus control signals are configured to their primary functions.

Address: 0xFC0A_4062 (PAR_FBCTL) Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_OE	PAR_TA	PAR_RWB	PAR_TS		0	0	0
W								
Reset:	1	1	1	1	1	0	0	0

Figure 16-36. FlexBus Control Pin Assignment Register (PAR_FBCTL)

Table 16-11. PAR_FBCTL Field Descriptions

Field	Description
7 PAR_OE	$\overline{\text{FB\_OE}}$ pin assignment. 0 $\overline{\text{FB\_OE}}$ pin configured for GPIO 1 $\overline{\text{FB\_OE}}$ pin configured for FlexBus output enable function
6 PAR_TA	$\overline{\text{FB\_TA}}$ pin assignment. 0 $\overline{\text{FB\_TA}}$ pin configured for GPIO 1 $\overline{\text{FB\_TA}}$ pin configured for FlexBus transfer acknowledge function
5 PAR_RWB	FB_R/$\overline{\text{W}}$ pin assignment. 0 FB_R/$\overline{\text{W}}$ pin configured for GPIO 1 FB_R/$\overline{\text{W}}$ pin configured for FlexBus read/write function
4–3 PAR_TS	$\overline{\text{FB\_TS}}$ pin assignment. 00 $\overline{\text{FB\_TS}}$ pin configured for GPIO 01 $\overline{\text{FB\_TS}}$ pin configured for FlexBus trsnsfer burst function 10 $\overline{\text{FB\_TS}}$ pin configured for FlexBus address latch enable function 11 $\overline{\text{FB\_TS}}$ pin configured for FlexBus transfer start function
2–0	Reserved, should be cleared.

16.3.5.4 DSPI Pin Assignment Register (PAR_DSPI)

The PAR_DSPI register controls the functions of the DSPI pins.

Address: 0xFC0A_4063 (PAR_DSPI) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PAR_PCS5	PAR_PCS2	PAR_PCS1	PAR_PCS0	PAR_SIN	PAR_SOUT	PAR_SCK
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-37. DSPI Pin Assignment Register (PAR_DSPI)

Table 16-12. PAR_DSPI Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 PAR_PCS5	DSPI_PCS5 pin assignment. 0 DSPI_PCS5 pin configured for GPIO 1 DSPI_PCS5 pin configured for DSPI peripheral chip select 5 function
5 PAR_PCS2	DSPI_PCS2 pin assignment. 0 DSPI_PCS2 pin configured for GPIO 1 DSPI_PCS2 pin configured for DSPI peripheral chip select 2 function
4 PAR_PCS1	DSPI_PCS1 pin assignment. 0 DSPI_PCS1 pin configured for GPIO 1 DSPI_PCS1 pin configured for DSPI peripheral chip select 1 function
3 PAR_PCS0	DSPI_PCS0 pin assignment. 0 DSPI_PCS0 pin configured for GPIO 1 DSPI_PCS0 pin configured for DSPI peripheral chip select 0 function

Table 16-12. PAR_DSPI Field Descriptions (continued)

Field	Description
2 PAR_SIN	DSPI_SIN pin assignment. 0 DSPI_SIN pin configured for GPIO 1 DSPI_SIN pin configured for DSPI serial data input function
1 PAR_SOUT	DSPI_SOUT pin assignment. 0 DSPI_SOUT pin configured for GPIO 1 DSPI_SOUT pin configured for DSPI serial data output function
0 PAR_SCK	DSPI_SCK pin assignment. 0 DSPI_SCK pin configured for GPIO 1 DSPI_SCK pin configured for DSPI serial clock function

16.3.5.5 Byte Enable Pin Assignment Register (PAR_BE)

The PAR_BE register controls the functions of the byte enable pins. After reset, the byte enable signals are configured to their primary functions.

Address: 0xFC0A_4064 (PAR_BE) Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_BE3		PAR_BE2			PAR_BE1		PAR_BE0
W					▨		▨	
Reset:	1	1	1	1	0	1	0	1

Figure 16-38. Byte Enable Pin Assignment Register (PAR_BE)

Table 16-13. PAR_BE Field Descriptions

Field	Description
7–6 PAR_BE3	FB_BE3 pin assignment. 00 FB_BE3 pin configured for GPIO 01 Reserved 10 FB_BE3 pin configured for FlexBus transfer size 1 function 11 FB_BE3 pin configured for FlexBus byte enable 3 function
5–4 PAR_BE2	FB_BE2 pin assignment. 00 FB_BE2 pin configured for GPIO 01 Reserved 10 FB_BE2 pin configured for FlexBus transfer size 0 function 11 FB_BE2 pin configured for FlexBus byte enable 2 function
3	Reserved, should be cleared.
2 PAR_BE1	FB_BE1 pin assignment. 0 FB_BE1 pin configured for GPIO 1 FB_BE1 pin configured for FlexBus byte enable 1 function
1	Reserved, should be cleared.
3–0 PAR_BE0	FB_BE0 pin assignment. 0 FB_BE0 pin configured for GPIO 1 FB_BE0 pin configured for FlexBus byte enable 0 function

16.3.5.6 Chip Select Pin Assignment Register (PAR_CS)

The PAR_CS register controls the functions of the FlexBus chip select pins. After reset, the byte enable signals are configured to their primary functions.

Address: 0xFC0A_4065 (PAR_CS)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_CS3	PAR_CS2	PAR_CS1	0
W								
Reset:	0	0	0	0	1	1	1	0

Figure 16-39. Chip Select Pin Assignment Register (PAR_CS)

Table 16-14. PAR_CS Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3 PAR_CS3	FB_CS3 pin assignment. 0 FB_CS3 pin configured for GPIO 1 FB_CS3 pin configured for FlexBus chip select 3 function
2 PAR_CS2	FB_CS2 pin assignment. 0 FB_CS2 pin configured for GPIO 1 FB_CS2 pin configured for FlexBus chip select 2 function
1 PAR_CS1	FB_CS1 pin assignment. 0 FB_CS1 pin configured for GPIO 1 FB_CS1 pin configured for FlexBus chip select 1 function
0	Reserved, should be cleared.

16.3.5.7 Timer Pin Assignment Registers (PAR_TIMER)

The PAR_TIMER register controls the functions of the DMA timer pins.

Address: 0xFC0A_4066 (PAR_TIMER)   Access: User read/write

	7	6	5	4	3	2	1	0
R W	PAR_T3IN		PAR_T2IN		PAR_T1IN		PAR_T0IN	
Reset:	0	0	0	0	0	0	0	0

Figure 16-40. Timer Pin Assignment (PAR_TIMER)

Table 16-15. PAR_TIMER Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–6
PAR_T3IN
5–4
PAR_T2IN
3–2
PAR_T1IN
1–0
PAR_T0IN</td><td>DMA timer pin assignment. These bit fields configure the DMA timer pins for one of their primary functions or GPIO.
<table>
<tr><th></th><th>PAR_T3IN</th><th>PAR_T2IN</th><th>PAR_T1IN</th><th>PAR_T0IN</th></tr>
<tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr>
<tr><td>01</td><td>U2RXD</td><td>U2TXD</td><td>$\overline{\text{U2CTS}}$</td><td>$\overline{\text{U2RTS}}$</td></tr>
<tr><td>10</td><td>T3OUT</td><td>T2OUT</td><td>T1OUT</td><td>T0OUT</td></tr>
<tr><td>11</td><td>T3IN</td><td>T2IN</td><td>T1IN</td><td>T0IN</td></tr>
</table>
</td></tr>
</table>

16.3.5.8 USB Pin Assignment Registers (PAR_USB)

The PAR_USB register controls the functions of the USB pins.

Address: 0xFC0A_4067 (PAR_USB) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_VBUSEN		PAR_VBUSOC	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 16-41. USB Pin Assignment (PAR_USB)

Table 16-16. PAR_USB Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–4</td><td>Reserved, should be cleared.</td></tr>
<tr><td>3–2
PAR_VBUSEN
1–0
PAR_VBUSOC</td><td>USB VBUS pin assignment. Configure the USB VBUS pins for one of their primary functions or GPIO.
<table>
<tr><th></th><th>PAR_VBUSEN</th><th>PAR_VBUSOC</th></tr>
<tr><td>00</td><td>GPIO</td><td>GPIO</td></tr>
<tr><td>01</td><td>ULPI_NXT</td><td>ULPI_STP</td></tr>
<tr><td>10</td><td>USB_PULLUP</td><td>Reserved</td></tr>
<tr><td>11</td><td>USB_VBUS_EN</td><td>USB_VBUS_OC</td></tr>
</table>
</td></tr>
</table>

16.3.5.9 UART Pin Assignment Register (PAR_UART)

The PAR_UART register controls the functions of the UART pins.

Address: 0xFC0A_4069 (PAR_UART) Access: User read/write

	7	6	5	4	3	2	1	0
R W	PAR_ U1CTS	PAR_ U1RTS	PAR_ U1RXD	PAR_ U1TXD	PAR_ U0CTS	PAR_ U0RTS	PAR_ U0RXD	PAR_ U0TXD
Reset:	0	0	0	0	0	0	0	0

Figure 16-42. UART Pin Assignment (PAR_UART)

Table 16-17. PAR_UART Field Descriptions

Field	Description
7 PAR_U1CTS	$\overline{\text{U1CTS}}$ pin assignment. 0 $\overline{\text{U1CTS}}$ pin configured for GPIO 1 $\overline{\text{U1CTS}}$ pin configured for UART1 clear-to-send function
6 PAR_U1RTS	$\overline{\text{U1RTS}}$ pin assignment. 0 $\overline{\text{U1RTS}}$ pin configured for GPIO 1 $\overline{\text{U1RTS}}$ pin configured for UART1 request-to-send function
5 PAR_U1RXD	U1RXD pin assignment. 0 U1RXD pin configured for GPIO 1 U1RXD pin configured for UART1 receive data function
4 PAR_U1TXD	U1TXD pin assignment. 0 U1TXD pin configured for GPIO 1 U1TXD pin configured for UART1 transmit data function
3 PAR_U0CTS	$\overline{\text{U0CTS}}$ pin assignment. 0 $\overline{\text{U0CTS}}$ pin configured for GPIO 1 $\overline{\text{U0CTS}}$ pin configured for UART0 clear-to-send function
2 PAR_U0RTS	$\overline{\text{U0RTS}}$ pin assignment. 0 $\overline{\text{U0RTS}}$ pin configured for GPIO 1 $\overline{\text{U0RTS}}$ pin configured for UART0 request-to-send function
1 PAR_U0RXD	U0RXD pin assignment. 0 U0RXD pin configured for GPIO 1 U0RXD pin configured for UART0 receive data function
0 PAR_U0TXD	U0TXD pin assignment. 0 U0TXD pin configured for GPIO 1 U0TXD pin configured for UART0 transmit data function

16.3.5.10 FEC/I2C Pin Assignment Register (PAR_FECI2C)

The PAR_FECI2C register controls the functions of the I^2C and the FEC MDC and MDIO pins.

Address: 0xFC0A_406A (PAR_FECI2C) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_MDC1		PAR_MDIO1		0	PAR_ MDC0	0	PAR_ MDIO0	PAR_SCL		PAR_SDA	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-43. FEC/I2C Pin Assignment (PAR_FECI2C)

Table 16-18. PAR_FECI2C Field Descriptions

Field	Description
15–12	Reserved, should be cleared.
11–10 PAR_MDC1 9–8 PAR_MDIO1	FEC1 MDIO and MDC pin assignment. These bit fields configure the FEC1_MDC and FEC1_MDIO pins for one of their primary functions or GPIO.
7	Reserved, should be cleared.
6 PAR_MDC0	FEC0 MDC pin assignment. 0 FEC0_MDC pin configured for GPIO 1 FEC0_MDC pin configured for FEC0 management data clock function
5	Reserved, should be cleared.
4 PAR_MDIO0	FEC0 MDIO pin assignment. 0 FEC0_MDIO pin configured for GPIO 1 FEC0_MDIO pin configured for FEC0 management data function
3–2 PAR_SCL 1–0 PAR_SDA	I2C_SCL and I2C_SDA pin assignment. These bit fields configure the I2C_SCL and I2C_SDA pins for one of their primary functions or GPIO.

	PAR_MDC1	**PAR_MDIO1**
00	GPIO	GPIO
01	Reserved	Reserved
10	ATA_DIOR	ATA_DIOW
11	FEC_MDC	FEC_MDIO

	PAR_SCL	**PAR_SDA**
00	GPIO	GPIO
01	U2TXD	U2RXD
10	Reserved	Reserved
11	I2C_SCL	I2C_SDA

16.3.5.11 SSI Pin Assignment Register (PAR_SSI)

The PAR_SSI register controls the functions of the SSI pins.

Address: 0xFC0A_406C (PAR_SSI) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PAR_BCLK		PAR_FS		PAR_RXD		PAR_TXD		0	PAR_MCLK
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-44. SSI Pin Assignment (PAR_SSI)

Table 16-19. PAR_SSI Field Descriptions

Field	Description
15–10	Reserved, should be cleared.
9–8 PAR_BCLK 7–6 PAR_FS 5–4 PAR_RXD 3–2 PAR_TXD	SSI pin assignment. Configure the SSI pins for one of their primary functions or GPIO.
1	Reserved, should be cleared.
0 PAR_MCLK	SSI_MCLK pin assignment. 0 SSI_MCLK pin configured for GPIO function. 1 SSI_MCLK pin configured for SSI oversampling clock function.

	PAR_BCLK	PAR_FS	PAR_RXD	PAR_TXD
00	GPIO	GPIO	GPIO	GPIO
01	Reserved	Reserved	Reserved	Reserved
10	$\overline{\text{U1CTS}}$	$\overline{\text{U1RTS}}$	U1RXD	U1TXD
11	SSI_BCLK	SSI_FS	SSI_RXD	SSI_TXD

16.3.5.12 ATA Pin Assignment Register (PAR_ATA)

The PAR_ATA register controls the functions of the ATA pins.

Address: 0xFC0A_406E (PAR_ATA) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	PAR_BUFEN	PAR_ACS1	PAR_ACS0	PAR_DA2	PAR_DA1	PAR_DA0	0	0	PAR_ARESET	PAR_DMARQ	PAR_IORDY
W																
(360 TEPBGA) Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
(256 MAPBGA) Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-45. ATA Pin Assignment (PAR_ATA)

Table 16-20. PAR_ATA Field Descriptions

Field	Description
15–11	Reserved, should be cleared.
10 PAR_BUFEN	ATA_BUFFER_EN pin assignment. 0 ATA_BUFFER_EN pin configured as GPIO. 1 ATA_BUFFER_EN pin configured for ATA buffer direction control function.
9 PAR_ACS1	$\overline{\text{ATA\_CS1}}$ pin assignment. 0 $\overline{\text{ATA\_CS1}}$ pin configured as GPIO. 1 $\overline{\text{ATA\_CS1}}$ pin configured for ATA chip select 1 function.
8 PAR_ACS0	$\overline{\text{ATA\_CS0}}$ pin assignment. 0 $\overline{\text{ATA\_CS0}}$ pin configured as GPIO. 1 $\overline{\text{ATA\_CS0}}$ pin configured for ATA chip select 0 function.
7 PAR_DA2	ATA_DA2 pin assignment. 0 ATA_DA2 pin configured as GPIO. 1 ATA_DA2 pin configured for ATA address 2 function.
6 PAR_DA1	ATA_DA1 pin assignment. 0 ATA_DA1 pin configured as GPIO. 1 ATA_DA1 pin configured for ATA address 1 function.
5 PAR_DA0	ATA_DA0 pin assignment. 0 ATA_DA0 pin configured as GPIO. 1 ATA_DA0 pin configured for ATA address 0 function.
4–3	Reserved, should be cleared.
2 PAR_ARESET	$\overline{\text{ATA\_RESET}}$ pin assignment. 0 $\overline{\text{ATA\_RESET}}$ pin configured as GPIO. 1 $\overline{\text{ATA\_RESET}}$ pin configured for ATA reset function.
1 PAR_DMARQ	ATA_DMARQ pin assignment. 0 ATA_DMARQ pin configured as GPIO. 1 ATA_DMARQ pin configured for ATA DMA request function.
0 PAR_IORDY	ATA_IORDY pin assignment. 0 ATA_IORDY pin configured as GPIO. 1 ATA_IORDY pin configured for ATA wait function.

16.3.5.13 IRQ Pin Assignment Register (PAR_IRQ)

The PAR_IRQ register controls the functions of the IRQ pins.

Address: 0xFC0A_4070 (PAR_IRQ) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PAR_IRQ4	0	0	PAR_IRQ1	0
W								
Reset:	0	0	0	0	0	0	See Note	0

Note: For 256-pin devices and 360-pin devices where the PCI is disabled through reset configuration, the PAR_IRQ reset state is 0. For 360-pin devices where PCI is enabled through reset configuration, the reset state is 1 when the PCI is an agent and 0 when the PCI is a host. Therefore, reset state is 0 when BOOTMOD[1:0] equals 00, and reset state is the complement value of FB_AD[3] when BOOTMOD[1:0] equals 10. For BOOTMOD equaling 11, reset state is the value of serial boot bit 103.

Figure 16-46. IRQ Pin Assignment (PAR_IRQ)

Table 16-21. PAR_IRQ Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4 PAR_IRQ4	$\overline{\text{IRQ4}}$ pin assignment. 0 $\overline{\text{IRQ4}}$ pin configured as GPIO or external interrupt request 4 function as determined by the edge port module. See Chapter 18, “Edge Port Module (EPORT),” for details. 1 $\overline{\text{IRQ4}}$ pin configured for SSI input clock fuction.
3–2	Reserved, should be cleared.
1 PAR_IRQ1	$\overline{\text{IRQ1}}$ pin assignment. 0 $\overline{\text{IRQ1}}$ pin configured as GPIO or external interrupt request 1 function as determined by the edge port module. See Chapter 18, “Edge Port Module (EPORT),” for details. 1 $\overline{\text{IRQ1}}$ pin configured for PCI interrupt fuction.
0	Reserved, should be cleared.

16.3.5.14 PCI Pin Assignment Register (PAR_PCI)

The PAR_PCI register controls the functions of the PCI grant and request pins.

Address: 0xFC0A_4072 (PAR_PCI) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PAR_GNT3		0	PAR_GNT2	0	PAR_GNT1	0	PAR_GNT0	PAR_REQ3		0	PAR_REQ2	0	PAR_REQ1	0	PAR_REQ0
W																
Reset	See Note		0	See Note	0	See Note	0	See Note	See Note		0	See Note	0	See Note	0	See Note

Note: Reset state is 1 when the PCI is enabled through reset configuration and is 0 otherwise.

Figure 16-47. PCI Pin Assignment (PAR_PCI)

Table 16-22. PAR_PCI Field Descriptions

Field	Description
15–14 PAR_GNT3	PCI_GNT3 assignment. Configure the PCI_GNT3 pin for one of its primary functions or GPIO. 00 PCI_GNT3 pin configured for GPIO 01 Reserved 10 PCI_GNT3 pin configured for ATA DMA acknowledge function 11 PCI_GNT3 pin configured for PCI grant 3 function
13	Reserved, should be cleared.
12 PAR_GNT2	PCI_GNT2 assignment. Configure the PCI_GNT2 pin for one of its primary functions or GPIO. 0 PCI_GNT2 pin configured for GPIO 1 PCI_GNT2 pin configured for PCI grant 2 function
11	Reserved, should be cleared.
10 PAR_GNT1	PCI_GNT1 assignment. Configure the PCI_GNT1 pin for one of its primary functions or GPIO. 0 PCI_GNT1 pin configured for GPIO 1 PCI_GNT1 pin configured for PCI grant 1 function
9	Reserved, should be cleared.
8 PAR_GNT0	PCI_GNT0 assignment. Configure the PCI_GNT0 pin for one of its primary functions or GPIO. 0 PCI_GNT0 pin configured for GPIO 1 PCI_GNT0 pin configured for PCI grant 0 function
7–6 PAR_REQ3	PCI_REQ3 assignment. Configure the PCI_REQ3 pin for one of its primary functions or GPIO. 00 PCI_REQ3 pin configured for GPIO 01 Reserved 10 PCI_REQ3 pin configured for ATA interrupt request function 11 PCI_REQ3 pin configured for PCI request 3 function
5	Reserved, should be cleared.
4 PAR_REQ2	PCI_REQ2 assignment. Configure the PCI_REQ2 pin for one of its primary functions or GPIO. 0 PCI_REQ2 pin configured for GPIO 1 PCI_REQ2 pin configured for PCI request 2 function
3	Reserved, should be cleared.
2 PAR_REQ1	PCI_REQ1 assignment. Configure the PCI_REQ1 pin for one of its primary functions or GPIO. 0 PCI_REQ1 pin configured for GPIO 1 PCI_REQ1 pin configured for PCI request 1 function
1	Reserved, should be cleared.
0 PAR_REQ0	PCI_REQ0 assignment. Configure the PCI_REQ0 pin for one of its primary functions or GPIO. 0 PCI_REQ0 pin configured for GPIO 1 PCI_REQ0 pin configured for PCI request 0 function

16.3.6 SDRAM Mode Select Control Register (MSCR_SDRAM)

The MSCR_SDRAM register controls the slew rate mode of the dedicated SDRAM pins.

Address: 0xFC0A_4074 (MSCR_SDRAM)

Access: User read/write

	7	6	5	4	3	2	1	0
R W	MSCR_SDDATA		MSCR_SDDQS		MSCR_SDCLK		MSCR_SDCTL	
Reset:	1	1	1	1	1	1	1	1

Figure 16-48. SDRAM Mode Select Control Register (MSCR_SDRAM)

Table 16-23. MSCR_SDRAM Field Descriptions

Field	Description
7–6 MSCR_ SDDATA	SD_D[31:16] slew rate mode. Controls the strength of the SDRAM upper data pins. 00 Half strength 1.8V mobile DDR 01 Full strength 1.8V mobile DDR 10 1.8V DDR2 without on-chip termination 11 2.5V DDR1
5–4 MSCR_ SDDQS	SD_DQS[3:2] slew rate mode. Controls the strength of the SDRAM DQS pins. 00 Half strength 1.8V mobile DDR 01 Full strength 1.8V mobile DDR 10 1.8V DDR2 without on-chip termination 11 2.5V DDR1
3–2 MSCR_ SDCLK	SD_CLK and $\overline{\text{SD\_CLK}}$ slew rate mode. Controls the strength of the SDRAM clock pins. 00 Half strength 1.8V mobile DDR 01 Full strength 1.8V mobile DDR 10 1.8V DDR2 without on-chip termination 11 2.5V DDR1
1–0 MSCR_ SDCTL	SD_A10, $\overline{\text{SD\_CAS}}$, SD_CKE, $\overline{\text{SD\_CS}}$[1:0], SD_DQS[3:2], $\overline{\text{SD\_RAS}}$, SD_WE slew rate mode. Controls the strength of the SDRAM control pins. 00 Half strength 1.8V mobile DDR 01 Full strength 1.8V mobile DDR 10 1.8V DDR2 without on-chip termination 11 2.5V DDR1

16.3.7 PCI Mode Select Control Register (MSCR_PCI)

The MSCR_PCI register controls the slew rate mode of the following PCI pins: PCI_AD[31:0], PCI_CBE[3:0], PCI_DEVSEL, PCI_FRAME, PCI_GNT[3:0], PCI_IDSEL, PCI_IRDY, PCI_PAR, PCI_PERR, PCI_REQ[3:0], PCI_RST, PCI_SERR, PCI_STOP, PCI_TRDY, and IRQ1.

Address: 0xFC0A_4075 (MSCR_PCI) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	MSCR_PCI
W								
Reset:	0	0	0	0	0	0	0	See Note

Note: Reset state is 1 when BOOTMOD[1:0] equals 00, the value of FB_AD[2] when BOOTMOD[1:0] equals 10, or the value of serial boot bit 104 when BOOTMOD[1:0] equals 11.

Figure 16-49. PCI Mode Select Control Register (MSCR_PCI)

Table 16-24. MSCR_PCI Field Descriptions

Field	Description
7–1	Reserved, should be cleared.
0 MSCR_PCI	PCI slew rate mode bit. Sets the slew rate mode for the PCI pins. 0 Low slew rate mote (33 MHz) 1 High slew rate mode (66 MHz)

16.3.8 Drive Strength Control Registers (DSCR_*x*)

The drive strength control registers set the output pin drive strengths. All drive strength control registers are read/write.

NOTE

These drive strength settings are effective in all non-JTAG modes, regardless of the current functions of the pins.

Address: 0xFC0A_4078 (DSCR_I2C)
0xFC0A_407C (DSCR_DSPI)
0xFC0A_407D (DSCR_TIMER)
0xFC0A_407E (DSCR_SSI)
0xFC0A_407F (DSCR_DMA)
0xFC0A_4080 (DSCR_DEBUG)
0xFC0A_4081 (DSCR_RESET)
0xFC0A_4082 (DSCR_IRQ)
0xFC0A_4083 (DSCR_USB)
0xFC0A_4084 (DSCR_ATA)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	DSE_*x*	
W								
Reset:	0	0	0	0	0	0	1	1

Figure 16-50. Drive Strength Control Registers (DSCR_*x*)

Address: 0xFC0A_407A (DSCR_FEC)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	DSE_FEC1		DSE_FEC0	
W								
Reset:	0	0	0	0	1	1	1	1

Figure 16-51. FEC Drive Strength Control Register (DSCR_FEC)

Address: 0xFC0A_407B (DSCR_UART)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	DSE_UART1		DSE_UART0	
W								
Reset:	0	0	0	0	1	1	1	1

Figure 16-52. UART Drive Strength Control Register (DSCR_UART)

Address: 0xFC0A_4079 (DSCR_FLEXBUS)

Access: User read/write

	7	6	5	4	3	2	1	0
R W	DSE_FBCLK		DSE_FBCTL		DSE_FBADH		DSE_FBADL	
Reset:	1	1	1	1	1	1	1	1

Figure 16-53. FlexBus Drive Strength Control Register (DSCR_FLEXBUS)

Table 16-25. DSCR_*x* Field Descriptions

Field	Description
DSCR_*x*	Drive strength control. Controls the drive strength of the various pins. See Table 16-26 for a list of the pins affected for each register bit field. 00 10pF 01 20pF 10 30pF 11 50pF

Note: See above figures for bit field positions.

Table 16-26. DSCR_*x* Pins Affected

Register (DSCR_*x*)	Pins Affected
DSCR_I2C	I2C_SDA and I2C_SCL
DSCR_FLEXBUS	See below for individual bit fields.
DSE_FBCLK	FB_CLK
DSE_FBCTL	$\overline{\text{FB\_OE}}$, FB_R/$\overline{\text{W}}$, $\overline{\text{FB\_TS}}$, $\overline{\text{FB\_TA}}$, $\overline{\text{FB\_BE/BWE}}$[3:0], and $\overline{\text{FB\_CS}}$[3:0]
DSE_FBADH	FB_AD[31:16]
DSE_FBADL	FB_AD[15:0]
DSCR_FEC	See below for individual bit fields.
DSE_FEC1	FEC1_MDC, FEC1_MDIO, FEC1_COL, FEC1_CRS, FEC1_RXCLK, FEC1_RXDV, FEC1_RXER, FEC1_RXD[3:0], FEC1_TXCLK, FEC1_TXD[3:0], FEC1_TXEN, and FEC1_TXER
DSE_FEC0	FEC0_MDC, FEC0_MDIO, FEC0_COL, FEC0_CRS, FEC0_RXCLK, FEC0_RXDV, FEC0_RXER, FEC0_RXD[3:0], FEC0_TXCLK, FEC0_TXD[3:0], FEC0_TXEN, and FEC0_TXER
DSCR_UART	See below for individual bit fields.
DSE_UART1	U1RXD, U1TXD, $\overline{\text{U1CTS}}$, and $\overline{\text{U1RTS}}$
DSE_UART0	U0RXD, U0TXD, $\overline{\text{U0CTS}}$, and $\overline{\text{U0RTS}}$
DSCR_DSPI	DSPI_PCS[5,2:0], DSPI_SCK, DSPI_SIN, and DSPI_SOUT
DSCR_TIMER	T3IN, T2IN, T1IN, and T0IN
DSCR_SSI	SSI_MCLK, SSI_BCLK, SSI_FS, SSI_RXD, and SSI_TXD
DSCR_DMA	$\overline{\text{DACK}}$[1:0], $\overline{\text{DREQ}}$[1:0]
DSCR_DEBUG	PSTDDATA[7:0] and TDO (when configured for the DSO function, JTAG_EN is negated).
DSCR_RESET	$\overline{\text{RSTOUT}}$
DSCR_IRQ	IRQ[7,4:3]
DSCR_USB	USB_VBUS_EN and USB_VBUS_OC
DSCR_ATA	ATA_BUFFER_EN, $\overline{\text{ATA\_CS}}$[1:0], ATA_DA[2:0], $\overline{\text{ATA\_RESET}}$, ATA_DMARQ, and ATA_IORDY

16.4 Functional Description

16.4.1 Overview

Initial pin function is determined during reset configuration. The pin assignment registers allow you to select among various primary functions and general purpose I/O after reset. Most pins are configured as GPIO by default. The notable exceptions to this are external bus control pins, address/data pins, and chip select pins. These pins are configured for their primary functions after reset.

Every GPIO pin is individually configurable as an input or an output via a data direction register (PDDR_*x*). Every GPIO port has an output data register (PODR_*x*) and a pin data register (PPDSDR_*x*) to monitor and control the state of its pins. Data written to a PODR_*x* register is stored and then driven to the corresponding port *x* pins configured as outputs.

Reading a PODR_*x* register returns the current state of the register regardless of the state of the corresponding pins. Reading a PPDSDR_*x* register returns the current state of the corresponding pins when configured as general purpose I/O, regardless of whether the pins are inputs or outputs.

Every GPIO port has a PPDSDR_*x* register and a clear register (PCLRR_*x*) for setting or clearing individual bits in the PODR_*x* register. Initial pin output drive strength is determined during reset configuration. The DSCR_*x* registers allow the pin drive strengths to be configured on a per-function basis after reset.

16.4.2 Port Digital I/O Timing

Input data on all pins configured as general purpose input is synchronized to the rising edge of FB_CLK, as shown in Figure 16-54.

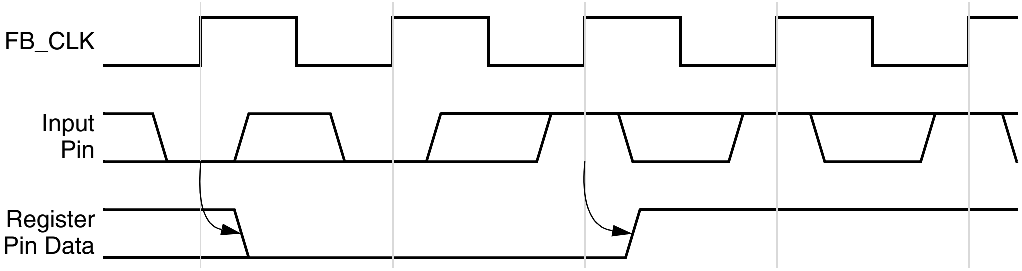


Figure 16-54. General Purpose Input Timing

Data written to the PODR_*x* register of any pin configured as a general purpose output is immediately driven to its respective pin, as shown in Figure 16-55.

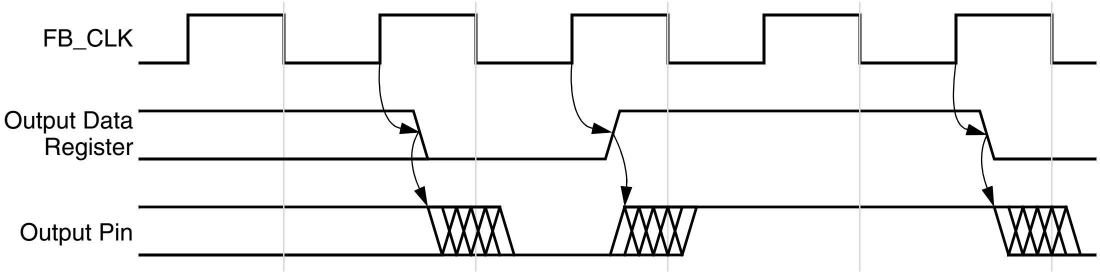


Figure 16-55. General Purpose Output Timing

16.5 Initialization/Application Information

The initialization for this module is done during reset configuration. All registers are reset to a predetermined state. Refer to Section 16.3, “Memory Map/Register Definition,” for more details on reset and initialization.

Chapter 17
Interrupt Controller Modules

17.1 Introduction

This section details the functionality of the interrupt controllers (INTC0, INTC1). The general features of the interrupt controller block include:

- 128 fully-programmable interrupt sources. Not all possible interrupt source locations are used on this device
- Each of the sources has a unique interrupt control register (ICR0*n*, ICR1*n*) to define the software-assigned levels
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source, plus global mask-all capability
- Supports hardware and software interrupt acknowledge cycles
- Wake-up signal from low-power stop modes

The 64, fully-programmable interrupt sources for the two interrupt controllers manage the complete set of interrupt sources from all of the modules on the device. This section describes how the interrupt sources are mapped to the interrupt controller logic and how interrupts are serviced.

17.1.1 68 K/ColdFire Interrupt Architecture Overview

Before continuing with the specifics of the interrupt controllers, a brief review of the interrupt architecture of the 68K/ColdFire family is appropriate.

The interrupt architecture of ColdFire is exactly the same as the M68000 family, where there is a 3-bit encoded interrupt priority level sent from the interrupt controller to the core, providing 7 levels of interrupt requests. Level 7 represents the highest priority interrupt level, while level 1 is the lowest priority. The processor samples for active interrupt requests once-per-instruction by comparing the encoded priority level against a 3-bit interrupt mask value (I) contained in bits 10:8 of the machine's status register (SR). If the priority level is greater than the SR[I] field at the sample point, the processor suspends normal instruction execution and initiates interrupt exception processing. Level 7 interrupts are treated as non-maskable and edge-sensitive within the processor, while levels 1-6 are treated as level-sensitive and may be masked depending on the value of the SR[I] field. For correct operation, the ColdFire device requires that, after asserted, the interrupt source remain asserted until explicitly disabled by the interrupt service routine.

During the interrupt exception processing, the CPU enters supervisor mode, disables trace mode, and then fetches an 8-bit vector from the interrupt controller. This byte-sized operand fetch is known as the interrupt acknowledge (IACK) cycle with the ColdFire implementation using a special memory-mapped address space within the interrupt controller. The fetched data provides an index into the exception vector table that contains 256 addresses, each pointing to the beginning of a specific exception service routine. In particular, vectors 64 - 255 of the exception vector table are reserved for user interrupt service routines. The first 64 exception vectors are reserved for the processor to manage reset, error conditions (access, address), arithmetic faults, system calls, etc. After the interrupt vector number has been retrieved, the processor continues by creating a stack frame in memory. For ColdFire, all exception stack frames are 2 longwords in length, and contain 32 bits of vector and status register data, along with the 32-bit program counter value of the instruction that was interrupted (see Section 3.3.3.1, "Exception Stack Frame Definition," for more information on the stack frame format). After the exception stack frame is stored in memory, the processor accesses the 32-bit pointer from the exception vector table using the vector number as the offset, and then jumps to that address to begin execution of the service routine. After the status register is stored in the exception stack frame, the SR[I] mask field is set to the level of the interrupt being acknowledged, effectively masking that level and all lower values while in the service routine.

The processing of the interrupt acknowledge cycle is fundamentally different than previous 68K/ColdFire cores. In this approach, all IACK cycles are directly managed by the interrupt controller, so the requesting peripheral device is not accessed during the IACK. As a result, the interrupt request must be explicitly cleared in the peripheral during the interrupt service routine. For more information, see Section 17.3.1.3, "Interrupt Vector Determination."

ColdFire processors guarantee that the first instruction of the service routine is executed before sampling for interrupts is resumed. By making this initial instruction a load of the SR, interrupts can be safely disabled, if required.

For more information on exception processing, see the *ColdFire Programmer's Reference Manual* at http://www.freescale.com/coldfire.

17.2 Memory Map/Register Definition

The register programming model for the interrupt controllers is memory-mapped to a 256-byte space. In the following discussion, there are a number of program-visible registers greater than 32 bits in size. For these control fields, the physical register is partitioned into two 32-bit values: a register high (the upper longword) and a register low (the lower longword). The nomenclature <reg_name>H and <reg_name>L is used to reference these values.

The registers and their locations are defined in Table 17-2. The base addresses for the interrupt controllers are listed below.

Table 17-1. Interrupt Controller Base Addresses

Interrupt Controller Number	Base Address
INTC0	0xFC04_8000
INTC1	0xFC04_C000
Global IACK Registers Space[1]	0xFC05_4000

[1] This address space only contains the global SWIACK and global L1ACK-L7IACK registers. See Section 17.2.10, "Software and Level 1–7 IACK Registers (SWIACKn, L1IACKn–L7IACKn)" for more information

Table 17-2. Interrupt Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
Interrupt Controller 0					
0xFC04_8000	Interrupt Pending Register High (IPRH0)	32	R	0x0000_0000	17.2.1/17-4
0xFC04_8004	Interrupt Pending Register Low (IPRL0)	32	R	0x0000_0000	17.2.1/17-4
0xFC04_8008	Interrupt Mask Register High (IMRH0)	32	R/W	0xFFFF_FFFF	17.2.2/17-5
0xFC04_800C	Interrupt Mask Register Low (IMRL0)	32	R/W	0xFFFF_FFFF	17.2.2/17-5
0xFC04_8010	Interrupt Force Register High (INTFRCH0)	32	R/W	0x0000_0000	17.2.3/17-6
0xFC04_8014	Interrupt Force Register Low (INTFRCL0)	32	R/W	0x0000_0000	17.2.3/17-6
0xFC04_801A	Interrupt Configuration Register (ICONFIG)	16	R/W	0x0000	17.2.4/17-7
0xFC04_801C	Set Interrupt Mask (SIMR0)	8	W	0x00	17.2.5/17-8
0xFC04_801D	Clear Interrupt Mask (CIMR0)	8	W	0x00	17.2.6/17-9
0xFC04_801E	Current Level Mask (CLMASK)	8	R/W	0x0F	17.2.7/17-9
0xFC04_801F	Saved Level Mask (SLMASK)	8	R/W	0x0F	17.2.8/17-10
0xFC04_8040 + *n* (*n*=0:63)	Interrupt Control Registers (ICR0*n*)	8	R/W	0x00	17.2.9/17-11
0xFC04_80E0	Software Interrupt Acknowledge (SWIACK0)	8	R	0x00	17.2.10/17-15
0xFC04_80E0 + 4*n* (*n*=1:7)	Level *n* Interrupt Acknowledge Registers (L*n*IACK0)	8	R	0x18	17.2.10/17-15
Interrupt Controller 1					
0xFC04_C000	Interrupt Pending Register High (IPRH1)	32	R	0x0000_0000	17.2.1/17-4
0xFC04_C004	Interrupt Pending Register Low (IPRL1)	32	R	0x0000_0000	17.2.1/17-4
0xFC04_C008	Interrupt Mask Register High (IMRH1)	32	R/W	0xFFFF_FFFF	17.2.2/17-5
0xFC04_C00C	Interrupt Mask Register Low (IMRL1)	32	R/W	0xFFFF_FFFF	17.2.2/17-5
0xFC04_C010	Interrupt Force Register High (INTFRCH1)	32	R/W	0x0000_0000	17.2.3/17-6
0xFC04_C014	Interrupt Force Register Low (INTFRCL1)	32	R/W	0x0000_0000	17.2.3/17-6
0xFC04_C01C	Set Interrupt Mask (SIMR1)	8	W	0x00	17.2.5/17-8
0xFC04_C01D	Clear Interrupt Mask (CIMR1)	8	W	0x00	17.2.5/17-8
0xFC04_C040 + *n* (*n*=1:63)	Interrupt Control Registers (ICR1*n*)	8	R/W	0x00	17.2.9/17-11
0xFC04_C0E0	Software Interrupt Acknowledge (SWIACK1)	8	R	0x00	17.2.10/17-15
0xFC04_C0E0 + 4*n* (*n*=1:7)	Level *n* Interrupt Acknowledge Registers (L*n*IACK1)	8	R	0x18	17.2.10/17-15
Global IACK Registers					

Table 17-2. Interrupt Controller Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC05_40E0	Global Software Interrupt Acknowledge (GSWIACK)	8	R	0x00	17.2.10/17-15
0xFC05_40E0 + 4*n* (*n*=1:7)	Global Level *n* Interrupt Acknowledge Registers (GL*n*IACK)	8	R	0x18	17.2.10/17-15

17.2.1 Interrupt Pending Registers (IPRH*n*, IPRL*n*)

The IPRH*n* and IPRL*n* registers, Figure 17-1 and Figure 17-2, are each 32 bits in size, and provide a bit map for each interrupt request to indicate if there is an active request (1 equals active request, 0 equals no request) for the given source. The interrupt mask register state does not affect the IPR*n*. The IPR*n* is cleared by reset and is a read-only register, so any attempted write to this register is ignored.

Address 0xFC04_8000 (IPRH0)
0xFC04_C000 (IPRH1)

Access: User 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
R	INT																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-1. Interrupt Pending Register High (IPRH*n*)

Table 17-3. IPRH*n* Field Descriptions

Field	Description
31–0 INT	Interrupt pending. Each bit corresponds to an interrupt source. The corresponding IMRH*n* bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRH*n* samples the signal generated by the interrupting source. The corresponding IPRH*n* bit reflects the state of the interrupt signal even if the corresponding IMRH*n* bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

Address 0xFC04_8004 (IPRL0)
0xFC04_C004 (IPRL1)

Access: User 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
R	INT																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-2. Interrupt Pending Register Low (IPRL*n*)

Table 17-4. IPRL*n* Field Descriptions

Field	Description
31–0 INT	Interrupt Pending. Each bit corresponds to an interrupt source. The corresponding IMRL*n* bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRL*n* samples the signal generated by the interrupting source. The corresponding IPRL*n* bit reflects the state of the interrupt signal even if the corresponding IMRL*n* bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

17.2.2 Interrupt Mask Register (IMRH*n*, IMRL*n*)

The IMRH*n* and IMRL*n* registers are each 32 bits in size and provide a bit map for each interrupt to allow the request to be disabled (1 equals disable the request, 0 equals enable the request). The IMRL register is used for masking interrupt sources 0 to 31, while the IMRH register is used for masking interrupts 32 to 63. The IMR*n* is set to all ones by reset, disabling all interrupt requests. The IMR*n* can be read and written.

NOTE

A spurious interrupt may occur if an interrupt source is being masked in the interrupt controller mask register (IMR) or a module’s interrupt mask register while the interrupt mask in the status register (SR[I]) is set to a value lower than the interrupt’s level. This is because by the time the status register acknowledges this interrupt, the interrupt has been masked. A spurious interrupt is generated because the CPU cannot determine the interrupt source. To avoid this situation for interrupts sources with levels 1-6, first write a higher level interrupt mask to the status register, before setting the mask in the IMR or the module’s interrupt mask register. After the mask is set, return the interrupt mask in the status register to its previous value. Because level 7 interrupts cannot be disabled in the status register prior to masking, use of the IMR or module interrupt mask registers to disable level 7 interrupts is not recommended.

Address 0xFC04_8008 (IMRH0)
0xFC04_C008 (IMRH1)

Access: User 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
R W	INT_MASK																															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 17-3. Interrupt Mask Register High (IMRH*n*)

Table 17-5. IMRH*n* Field Descriptions

Field	Description
31–0 INT_MASK	Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRH*n* bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRH*n* bit reflects the state of the interrupt signal even if the corresponding IMRH*n* bit is set. 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked

Address 0xFC04_800C (IMRL0)
0xFC04_C00C (IMRL1)

Access: User 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
R W	INT_MASK																															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 17-4. Interrupt Mask Register Low (IMRL*n*)

Table 17-6. IMRL*n* Field Descriptions

Field	Description
31–0 INT_MASK	Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRL*n* bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRL*n* bit reflects the state of the interrupt signal even if the corresponding IMRL*n* bit is set. 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked

17.2.3 Interrupt Force Registers (INTFRCH*n*, INTFRCL*n*)

The INTFRCH*n* and INTFRCL*n* registers are each 32 bits in size and provide a mechanism to allow software generation of interrupts for each possible source for functional or debug purposes. The system design may reserve one or more sources to allow software to self-schedule interrupts by forcing one or more of these bits (set to force request, clear to negate request) in the appropriate INTFRC*n* register. The INTFRCL*n* register forces interrupts for sources 0 to 31, while the INTFRCH*n* register forces interrupts for sources 32 to 63. The assertion of an interrupt request via the interrupt force register is not affected by the interrupt mask register. The INTFRC*n* registers are cleared by reset.

Address 0xFC04_8010 (INTFRCH0)
0xFC04_C010 (INTFRCH1)

Access: User 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
R W	INTFRCH																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-5. Interrupt Force Register High (INTFRCH*n*)

Table 17-7. INTFRCH*n* Field Descriptions

Field	Description
31–0 INTFRCH	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on the corresponding interrupt source 1 Force an interrupt on the corresponding source

Address 0xFC04_8014 (INTFRCL0)
0xFC04_C014 (INTFRCL1)

Access: User 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
R	INTFRCL																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-6. Interrupt Force Register Low (INTFRCL*n*)

Table 17-8. INTFRCL*n* Field Descriptions

Field	Description
31–0 INTFRCL	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on corresponding interrupt source 1 Force an interrupt on the corresponding source

17.2.4 Interrupt Configuration Register (ICONFIG)

This 16-bit register defines the operating configuration for the interrupt controller module.

NOTE

Only one copy of this register exists among the 2 interrupt controller modules. All reads and writes to this register must be made to the INTC0 memory space.

Address: 0xFC04_801A (ICONFIG)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ELVLPRI							0	0	0	EMASK	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-7. Interrupt Configuration Register (ICONFIG)

Table 17-9. ICONFIG Field Descriptions

Field	Description
15–9 ELVLPRI	Enable core's priority elevation on priority levels. Each ELVLPRI[7:1] bit corresponds to the available priority levels 1 – 7. If set, the assertion of the corresponding level-*n* request to the core causes the processor's bus master priority to be temporarily elevated in the device's crossbar switch arbitration logic. The processor's bus master arbitration priority remains elevated until the level-*n* request is negated. If round-robin arbitration is enabled, this bit has no effect. If cleared, the assertion of a level-n request does not affect the processor's bus master priority.
8–6	Reserved, must be cleared.
5 EMASK	If set, the interrupt controller automatically loads the level of an interrupt request into the CLMASK (current level mask) when the acknowledge is performed. At the exact same cycle, the value of the current interrupt level mask is saved in the SLMASK (saved level mask) register. This feature can be used to support software-managed nested interrupts, and is intended to complement the interrupt masking functions supported in the ColdFire processor. The value of SLMASK register should be read from the interrupt controller and saved in the interrupt stack frame in memory, and restored near the service routine's exit. If cleared, the INTC does not perform any automatic masking of interrupt levels. The state of this bit does not affect the ColdFire processor's interrupt masking logic in any manner.
4–0	Reserved, must be cleared.

17.2.5 Set Interrupt Mask Register (SIMR*n*)

The SIMR*n* register provides a simple mechanism to set a given bit in the IMR*n* registers to mask the corresponding interrupt request. The value written to the SIMR field causes the corresponding bit in the IMR*n* register to be set. The SIMR*n*[SALL] bit provides a global set function, forcing the entire contents of IMR*n* to be set, thus masking all interrupts. Reads of this register return all zeroes. This register is provided so interrupt service routines can easily mask the given interrupt request without the need to perform a read-modify-write sequence on the IMR*n* register.

Address: 0xFC04_801C (SIMR0)
0xFC04_C01C (SIMR1)

Access: User write-only

<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>W</td><td></td><td>SALL</td><td colspan="6">SIMR</td></tr>
<tr><td>Reset:</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 17-8. Set Interrupt Mask Register (SIMR*n*)

Table 17-10. SIMR*n* Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SALL	Set all bits in the IMR*n* register, masking all interrupt requests. 0 Only set those bits specified in the SIMR field. 1 Set all bits in IMR*n* register. The SIMR field is ignored.
5–0 SIMR	Set the corresponding bit in the IMR*n* register, masking the interrupt request.

17.2.6 Clear Interrupt Mask Register (CIMR*n*)

The CIMR*n* register provides a simple mechanism to clear a given bit in the IMR*n* registers to enable the corresponding interrupt request. The value written to the CIMR field causes the corresponding bit in the IMR*n* register to be cleared. The CIMR*n*[CALL] bit provides a global clear function, forcing the entire contents of IMR*n* to be cleared, thus enabling all interrupts. Reads of this register return all zeroes. This register is provided so interrupt service routines can easily enable the given interrupt request without the need to perform a read-modify-write sequence on the IMR*n* register.

In the event of a simultaneous write to the CIMR*n* and SIMR*n*, the SIMR*n* has priority and the resulting function would be a set of the interrupt mask register.

Address: 0xFC04_801D (CIMR0)
0xFC04_C01D (CIMR1)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CALL	CIMR					
Reset:	0	0	0	0	0	0	0	0

Figure 17-9. Clear Interrupt Mask Register (CIMR*n*)

Table 17-11. CIMR*n* Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CALL	Clear all bits in the IMR*n* register, enabling all interrupt requests. 0 Only set those bits specified in the CIMR field. 1 Clear all bits in IMR*n* register. The CIMR field is ignored.
5–0 CIMR	Clear the corresponding bit in the IMR*n* register, enabling the interrupt request.

17.2.7 Current Level Mask Register (CLMASK)

The CLMASK register is provided so the interrupt controller can optionally automatically manage masking of interrupt requests based on the programmed priority level. If enabled by ICONFIG[EMASK] bit being set, an interrupt acknowledge read cycle returns a vector number identifying the physical request source, and the CLMASK register is loaded with the level number associated with the request. After the CLMASK register is updated, then all interrupt requests with level numbers equal to or less than this value are masked by the controller and are not allowed to cause the assertion of the interrupt signal to the processor core. As the CLMASK register is updated during the IACK cycle read, the former value is saved in the SLMASK register.

Typically, after a level-*n* interrupt request is managed, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests. In addition, an interrupt service routine can explicitly load this register with a lower priority value to query for any pending interrupts via software interrupt acknowledge cycles.

NOTE

Only one copy of this register exists among the 2 interrupt controller modules. All reads and writes to this register must be made to the INTC0 memory space.

Address: 0xFC04_801E (CLMASK) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	CLMASK			
W								
Reset:	0	0	0	0	1	1	1	1

Figure 17-10. Current Level Mask Register (CLMASK)

Table 17-12. CLMASK Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3–0 CLMASK	Current level mask. Defines the level mask, where only interrupt levels greater than the current value are processed by the controller 0000 Level 1 – 7 requests are processed. 0001 Level 2 – 7 requests are processed. 0010 Level 3 – 7 requests are processed. 0011 Level 4 – 7 requests are processed. 0100 Level 5 – 7 requests are processed. 0101 Level 6 – 7 requests are processed. 0110 Level 7 requests are processed. 0111 All requests are masked. 1000 – 1110 Reserved. 1111 Level 1 – 7 requests are processed.

17.2.8 Saved Level Mask Register (SLMASK)

The SLMASK register is provided so the interrupt controller can optionally automatically manage masking of interrupt requests based on the programmed priority level. If enabled by ICONFIG[EMASK] bit being set, an interrupt acknowledge read cycle returns a vector number identifying the physical request source, and the CLMASK register is loaded with the level number associated with the request. After the CLMASK register is updated, then all interrupt requests with level numbers equal to or less than this value are masked by the controller and are not allowed to cause the assertion of the interrupt signal to the processor core. As the CLMASK register is updated during the IACK cycle read, the former value is saved in the SLMASK register.

Typically, after a level-*n* interrupt request is managed, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests.

NOTE

Only one copy of this register exists among the two interrupt controller modules. All reads and writes to this register must be made to the INTC0 memory space.

Address: 0xFC04_801F (SLMASK) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	SLMASK			
W								
Reset:	0	0	0	0	1	1	1	1

Figure 17-11. Saved Level Mask Register (SLMASK)

Table 17-13. SLMASK Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3–0 SLMASK	Saved level mask. Defines the saved level mask. See the CLMASK field definition for more information on the specific values.

17.2.9 Interrupt Control Register (ICR0*n*, ICR1*n*, (*n* = 00, 01, 02, ..., 63))

Each ICR register specifies the interrupt level (1–7) for the corresponding interrupt source. These registers are cleared by reset and should be programmed with the appropriate levels before interrupts are enabled.

When multiple interrupt requests are programmed to the same level number, they are processed in a descending request number order. As an example, if requests 63, 62, 2, and 1 are programmed to a common level, request 63 is processed first, then request 62, then request 2, and finally request 1.

This definition allows software maximum flexibility in grouping interrupt request sources within any given priority level. The priority level in the ICRs directly corresponds to the interrupt level supported by the ColdFire processor.

Address: 0xFC04_8040 – 7F (ICR000 – ICR063)
0xFC04_C040 – 7F (ICR100 – ICR163) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	LEVEL		
W								
Reset:	0	0	0	0	0	0	0	0

Figure 17-12. Interrupt Control Registers (ICR0*n*, ICR1*n*)

Table 17-14. ICR*n* Field Descriptions

Field	Description
7–3	Reserved, must be cleared.
2–0 LEVEL	Interrupt level. Indicates the interrupt level assigned to each interrupt input. A level of 0 effectively disables the interrupt request, while a level 7 interrupt is given the highest priority. If interrupt masking is enabled (ICONFIG[EMASK] = 1), the acknowledgement of a level-*n* request forces the controller to automatically mask all interrupt requests of level-*n* and lower.

17.2.9.1 Interrupt Sources

Table 17-15 and Table 17-16 list the interrupt sources for each interrupt request line for INTC0 and INTC1.

Table 17-15. Interrupt Source Assignment For INTC0

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0	Not Used			
1	EPORT	EPFR[EPF1]	Edge port flag 1	Write EPF1 = 1
2		EPFR[EPF2]	Edge port flag 2	Write EPF2 = 1
3		EPFR[EPF3]	Edge port flag 3	Write EPF3 = 1
4		EPFR[EPF4]	Edge port flag 4	Write EPF4 = 1
5		EPFR[EPF5]	Edge port flag 5	Write EPF5 = 1
6		EPFR[EPF6]	Edge port flag 6	Write EPF6 = 1
7		EPFR[EPF7]	Edge port flag 7	Write EPF7 = 1
8	DMA	EDMA_INTR[INT00]	DMA Channel 0 transfer complete	Write EDMA_CINTR[CINT] = 0
9		EDMA_INTR[INT01]	DMA Channel 1 transfer complete	Write EDMA_CINTR[CINT] = 1
10		EDMA_INTR[INT02]	DMA Channel 2 transfer complete	Write EDMA_CINTR[CINT] = 2
11		EDMA_INTR[INT03]	DMA Channel 3 transfer complete	Write EDMA_CINTR[CINT] = 3
12		EDMA_INTR[INT04]	DMA Channel 4transfer complete	Write EDMA_CINTR[CINT] = 4
13		EDMA_INTR[INT05]	DMA Channel 5 transfer complete	Write EDMA_CINTR[CINT] = 5
14		EDMA_INTR[INT06]	DMA Channel 6 transfer complete	Write EDMA_CINTR[CINT] = 6
15		EDMA_INTR[INT07]	DMA Channel 7 transfer complete	Write EDMA_CINTR[CINT] = 7
16		EDMA_INTR[INT08]	DMA Channel 8 transfer complete	Write EDMA_CINTR[CINT] = 8
17		EDMA_INTR[INT09]	DMA Channel 9 transfer complete	Write EDMA_CINTR[CINT] = 9
18		EDMA_INTR[INT10]	DMA Channel 10 transfer complete	Write EDMA_CINTR[CINT] = 10
19		EDMA_INTR[INT11]	DMA Channel 11 transfer complete	Write EDMA_CINTR[CINT] = 11
20		EDMA_INTR[INT12]	DMA Channel 12 transfer complete	Write EDMA_CINTR[CINT] = 12
21		EDMA_INTR[INT13]	DMA Channel 13 transfer complete	Write EDMA_CINTR[CINT] = 13
22		EDMA_INTR[INT14]	DMA Channel 14 transfer complete	Write EDMA_CINTR[CINT] = 14
23		EDMA_INTR[INT15]	DMA Channel 15 transfer complete	Write EDMA_CINTR[CINT] = 15
24		EDMA_ERR[ERR*n*]	DMA Error Interrupt	Write EDMA_CERR[CERR] = *n*
25	SCM	SCMIR[CWIC]	Core Watchdog Timeout	Write SCMISR[CWIC] = 1
26	UART0	UISR0 register	UART0 Interrupt Request	Automatically cleared
27	UART1	UISR1 register	UART1 Interrupt Request	Automatically cleared
28	UART2	UISR2 register	UART2 Interrupt Request	Automatically cleared
29	Not Used			

Table 17-15. Interrupt Source Assignment For INTC0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
30	I^2C	I2SR[IIF]	I^2C Interrupt	Write I2SR[IIF] = 0
31	DSPI	DSPI_SR register	DSPI interrupt (Logical OR of INTC1's source #33–39)	Write 1 to appropriate DSPI_SR bit
32	DTIM0	DTER0 register	Timer 0 interrupt	Write 1 to appropriate DTER0 bit
33	DTIM1	DTER1 register	Timer 1 interrupt	Write 1 to appropriate DTER1 bit
34	DTIM2	DTER2 register	Timer 2 interrupt	Write 1 to appropriate DTER2 bit
35	DTIM3	DTER3 register	Timer 3 interrupt	Write 1 to appropriate DTER3 bit
36	FEC0	EIR0[TXF]	Transmit frame interrupt	Write EIR0[TXF] = 1
37		EIR0[TXB]	Transmit buffer interrupt	Write EIR0[TXB] = 1
38		EIR0[UN]	Transmit FIFO underrun	Write EIR0[UN] = 1
39		EIR0[RL]	Collision retry limit	Write EIR0[RL] = 1
40		EIR0[RXF]	Receive frame interrupt	Write EIR0[RXF] = 1
41		EIR0[RXB]	Receive buffer interrupt	Write EIR0[RXB] = 1
42		EIR0[MII]	MII interrupt	Write EIR0[MII] = 1
43		EIR0[LC]	Late collision	Write EIR0[LC] = 1
44		EIR0[HBERR]	Heartbeat error	Write EIR0[HBERR] = 1
45		EIR0[GRA]	Graceful stop complete	Write EIR0[GRA] = 1
46		EIR0[EBERR]	Ethernet bus error	Write EIR0[EBERR] = 1
47		EIR0[BABT]	Babbling transmit error	Write EIR0[BABT] = 1
48		EIR0[BABR]	Babbling receive error	Write EIR0[BABR] = 1
49	FEC1	EIR1[TXF]	Transmit frame interrupt	Write EIR1[TXF] = 1
50		EIR1[TXB]	Transmit buffer interrupt	Write EIR1[TXB] = 1
51		EIR1[UN]	Transmit FIFO underrun	Write EIR1[UN] = 1
52		EIR1[RL]	Collision retry limit	Write EIR1[RL] = 1
53		EIR1[RXF]	Receive frame interrupt	Write EIR1[RXF] = 1
54		EIR1[RXB]	Receive buffer interrupt	Write EIR1[RXB] = 1
55		EIR1[MII]	MII interrupt	Write EIR1[MII] = 1
56		EIR1[LC]	Late collision	Write EIR1[LC] = 1
57		EIR1[HBERR]	Heartbeat error	Write EIR1[HBERR] = 1
58		EIR1[GRA]	Graceful stop complete	Write EIR1[GRA] = 1
59		EIR1[EBERR]	Ethernet bus error	Write EIR1[EBERR] = 1
60		EIR1[BABT]	Babbling transmit error	Write EIR1[BABT] = 1
61		EIR1[BABR]	Babbling receive error	Write EIR1[BABR] = 1

Table 17-15. Interrupt Source Assignment For INTC0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
62	SCM	SCMIR[CFEI]	Core bus error interrupt	Write SCMIR[CFEI] = 1
63	RTC	RTC_ISR	Real Time Clock Interrupt	Write one to corresponding bit in RTC_ISR.

Table 17-16. Interrupt Source Assignment for INTC1

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0–32	Not Used			
33	DSPI	DSPI_SR[EOQF]	End of queue interrupt	Write 1 to EOQF.
34		DSPI_SR[TFFF]	Transmit FIFO fill interrupt	Write 1 to TFFF.
35		DSPI_SR[TCF]	Transfer complete interrupt	Write 1 to TCF.
36		DSPI_SR[TFUF]	Transmit FIFO underflow interrupt	Write 1 to TFUF.
37		DSPI_SR[RFDF]	Receive FIFO not empty interrupt	Write 1 to RFDF after reading the DSPI_POPR register.
38		DSPI_SR[RFOF]	Receive FIFO overflow interrupt	Write 1 to RFOF.
39		DSPI_SR[RFOF] or DSPI_SR[TFUF]	Receive FIFO overflow or transmit FIFO underflow interrupt	Write 1 to either RFOF or TFUF.
40	RNG	EI	RNG interrupt flag	Write RNGCR[CI] = 1
41–42	Not Used			
43	PIT0	PCSR0[PIF]	PIT interrupt flag	Write PIF = 1 or write PMR
44	PIT1	PCSR1[PIF]	PIT interrupt flag	Write PIF = 1 or write PMR
45	PIT2	PCSR2[PIF]	PIT interrupt flag	Write PIF = 1 or write PMR
46	PIT3	PCSR3[PIF]	PIT interrupt flag	Write PIF = 1 or write PMR
47	USB OTG	USB_STS	USB OTG interrupt	Write 1 to corresponding bit in the USB_STS.
48	Not Used			
49	SSI	SSI_ISR	SSI interrupt	Various, see chapter for details.
50–52	Not Used			
53	CCM	UOCSR	USB status Interrupt	Read UOCSR.
54	ATA	ATA_ISR	ATA interrupt	Write a 1 to the necessary bit in ATA_ICR.
55	PCI	PCIGSCR	PCI interrupt	Write a 1 to the necessary bit in PCIGSCR.
56	PCI	PASR	PCI arbiter interrupt	Write a 1 to the necessary bit in PASR or to PACR[RA].
57	PLL	PSR[LOCKS]	PLL loss-of-lock interrupt	Wrtie a 0 to PSR[LOCKS]
58–63	Not Used			

17.2.10 Software and Level 1–7 IACK Registers (SWIACK*n*, L1IACK*n*–L7IACK*n*)

The eight IACK registers (per interrupt controller) can be explicitly addressed via the CPU, or implicitly addressed via a processor-generated interrupt acknowledge cycle during exception processing. In either case, the interrupt controller's actions are very similar.

First, consider an IACK cycle to a specific level: a level-*n* IACK. When this type of IACK arrives in the interrupt controller, the controller examines all the currently-active level *n* interrupt requests, determines the highest priority within the level, and then responds with the unique vector number corresponding to that specific interrupt source. The vector number is supplied as the data for the byte-sized IACK read cycle. In addition to providing the vector number, the interrupt controller also loads the level into the CLMASK register, where it may be retrieved later.

This interrupt controller design also supports the concept of a software IACK. A software IACK allows an interrupt service routine to determine if there are other pending interrupts so that the overhead associated with interrupt exception processing (including machine state save/restore functions) can be minimized. In general, the software IACK is performed near the end of an interrupt service routine, and if there are additional active interrupt sources, the current interrupt service routine (ISR) passes control to the appropriate service routine, but without taking another interrupt exception.

When the interrupt controller receives a software IACK read, it returns the vector number associated with the highest unmasked interrupt source for that interrupt controller. If there are no active sources, the interrupt controller returns an all-zero vector as the operand for the SWIACK register. A read from the L*n*IACK registers when there are no active requests returns a value of 24 (0x18), signaling a spurious interrupt.

In addition to the software IACK registers in each interrupt controller, there are global software IACK registers. A read from the global SWIACK (GSWIACK) returns the vector number for the highest level and priority unmasked interrupt source from all interrupt controllers. A read from one of the global L*n*IACK (GL*n*IACK) registers returns the vector for the highest priority unmasked interrupt within a level for all interrupt controllers.

Address: 0xFC04_80E0 (SWIACK0)
0xFC04_80E0+4*n* (L*n*IACK0) *n*=1:7
0xFC04_C0E0 (SWIACK1)
0xFC04_C0E0+4*n* (L*n*IACK1) *n*=1:7
0xFC05_40E0 (GSWIACK)
0xFC05_40E0+4*n* (GL*n*IACK) *n*=1:7

Access: User read-only

	7	6	5	4	3	2	1	0
R	VECTOR							
W								
Reset (SWIACK*n*):	0	0	0	0	0	0	0	0
Reset (L*n*IACK*n*):	0	0	0	1	1	0	0	0

Figure 17-13. Software and Level *n* IACK Registers (SWIACK*n*, L1IACK*n* – L7IACK*n*)

Table 17-17. SWIACK*n* and L*x*IACK*n* Field Descriptions

Field	Description
7–0 VECTOR	Vector number. A read from the SWIACK register returns the vector number associated with the highest priority pending interrupt source. A read from one of the L*n*IACK registers returns the highest priority unmasked interrupt source within the level. A write to any IACK register causes an error termination.

17.3 Functional Description

17.3.1 Interrupt Controller Theory of Operation

To support the interrupt architecture of the 68K/ColdFire programming model, the 64 interrupt sources are organized as 7 levels, with an arbitrary number of requests programmed to each level. The priority structure within a single interrupt level depends on the interrupt source number assignments (see Section 17.2.9.1, “Interrupt Sources”). The higher numbered interrupt source has priority over the lower numbered interrupt source. See the below table for an example.

Table 17-18. Example Interrupt Priority Within a Level

Interrupt Source	ICR[2:0]	Priority
40	011	Highest
22	011	
8	011	
2	011	Lowest

The level is fully programmable for all sources. The 3-bit level is defined in the interrupt control register (ICR0*n*, ICR1*n*).

The operation of the interrupt controller can be broadly partitioned into three activities:

- Recognition
- Prioritization
- Vector determination during IACK

17.3.1.1 Interrupt Recognition

The interrupt controller continuously examines the request sources (IPR*n*) and the interrupt mask register (IMR*n*) to determine if there are active requests. This is the recognition phase. The interrupt force register (INTFRC*n*) also factors into the generation of an active request.

17.3.1.2 Interrupt Prioritization

As an active request is detected, it is translated into the programmed interrupt level. Next, the appropriate level masking is performed if this feature is enabled. The level of the active request must be greater than the current mask level before it is signaled in the processor. The resulting unmasked decoded priority level is driven out of the interrupt controller. The decoded priority levels from the interrupt controllers are logically summed together, and the highest enabled interrupt request is sent to the processor core during this prioritization phase.

17.3.1.3 Interrupt Vector Determination

After the core has sampled for pending interrupts and begun interrupt exception processing, it generates an interrupt acknowledge cycle (IACK). The IACK transfer is treated as a memory-mapped byte read by the processor, and routed to the appropriate interrupt controller. Next, the interrupt controller extracts the level being acknowledged from address bits[4:2], and then determines the highest unmasked level for the type of interrupt being acknowledged, and returns the 8-bit interrupt vector for that request to complete the cycle. The 8-bit interrupt vector is formed using the following algorithm:

```
For INTC0,                  vector_number = 64 + interrupt source number
For INTC1,                  vector_number = 128 + interrupt source number
```

Recall vector_numbers 0-63 are reserved for the ColdFire processor and its internal exceptions. Thus, the following mapping of bit positions to vector numbers applies for INTC0:

```
if interrupt source 0 is active and acknowledged, then vector_number =  64
if interrupt source 1 is active and acknowledged, then vector_number =  65
if interrupt source 2 is active and acknowledged, then vector_number =  66
...
if interrupt source 63 is active and acknowledged, then vector_number = 127
```

The net effect is a fixed mapping between the bit position within the source to the actual interrupt vector number.

If there is no active interrupt source for the given level, a special spurious interrupt vector (vector_number equals 24) is returned and it is the responsibility of the service routine to manage this error situation.

This protocol implies the interrupting peripheral is not accessed during the acknowledge cycle because the interrupt controller completely services the acknowledge. This means the interrupt source must be explicitly disabled in the interrupt service routine. This design provides unique vector capability for all interrupt requests, regardless of the complexity of the peripheral device.

In some applications, it is expected that the hardware masking of interrupt levels by the interrupt controller is enabled. This masking capability can be used with the processor’s masking logic to form a dual-mask capability. In this operation mode, the IACK read cycle also causes the current interrupt level mask to be saved in the SLMASK register, and the new level being acknowledged loaded into the CLMASK register. This operation then automatically masks the new level (and all lower levels) while in the service routine. Generally, as the service routine completes execution, and the initiating request source has been negated, the saved mask level is restored into the current mask level to re-enable the lower priority levels.

Finally, the vector number returned during the IACK cycle provides the association with the request and the physical interrupt signal. The CLMASK and SLMASK registers are all loaded (if properly enabled) during the interrupt acknowledge read cycle.

17.3.2 Prioritization Between Interrupt Controllers

The interrupt controllers have a fixed priority, where INTC0 has the highest priority, and INTC1 has the lowest priority. If both interrupt controllers have active interrupts at the same level, then the INTC0 interrupt is serviced first. If INTC1 has an active interrupt with a higher level than the highest INTC0 interrupt, then the INTC1 interrupt is serviced first.

17.3.3 Low-Power Wake-up Operation

The system control module (SCM) contains an 8-bit low-power control register (LPCR) to control the low-power stop mode. This register must be explicitly programmed by software to enter low-power mode. It also contains a wake-up control register (WCR) sets the priority level of the interrupt necessary to bring the device out of the specified low-power mode. Refer to Chapter 9, "Power Management," for definitions of the LPCR and WCR registers, as well as more information on low-power modes.

Each interrupt controller provides a special combinatorial logic path to provide a special wake-up signal to exit from the low-power stop mode. This special mode of operation works as follows:

1. The WCR register is programmed, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. After the processor has stopped execution, it asserts a specific processor status (PST) encoding. Issuing the STOP instruction when the WCR[ENBWCR] bit is set causes the SCM to enter the mode specified in LPCR[LPMD].
3. The entry into a low-power mode is processed by the low-power mode control logic, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate a level 7 interrupt request or an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. After an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.
6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

For more information, see Section 9.2.1, "Wake-up Control Register (WCR)".

17.4 Initialization/Application Information

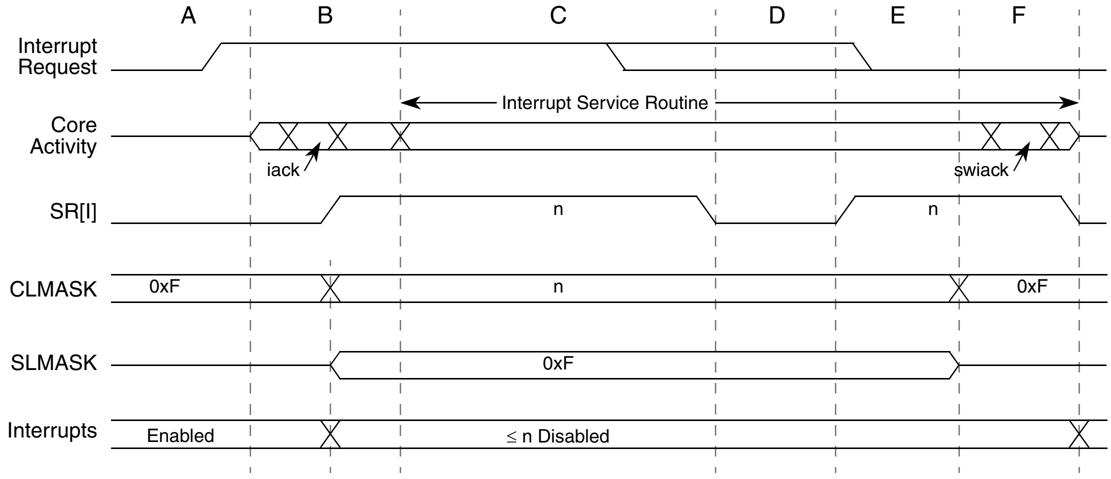
The interrupt controller's reset state has all requests masked via the IMR. Before any interrupt requests are enabled, the following steps must be taken:

1. Set the ICONFIG register to the desired system configuration.
2. Program the ICR*n* registers with the appropriate interrupt levels.
3. The reset value for the level mask registers (CLMASK and SLMASK) is 0xF (no levels masked). Typically, these registers do not need to be modified before interrupts are enabled.
4. Load the appropriate interrupt vector tables and interrupt service routines into memory.

5. Enable the interrupt requests, by clearing the appropriate bits in the IMR and lowering the interrupt mask level in the core's status register (SR[I]) to an appropriate level.

17.4.1 Interrupt Service Routines

This section focuses on the interaction of the interrupt masking functionality with the service routine. Figure 17-14 presents a timing diagram showing various phases during the execution of an interrupt service routine with the controller level masking functionality enabled. The time scale in this diagram is not meant to be accurate.



Note: Not to scale

Figure 17-14. Interrupt Service Routine and Masking

Consider the events depicted in each segment (A – F) of the above diagram.

In A, an interrupt request is asserted, which is then signalled to the core.

As B begins, the interrupt request is recognized, and the core begins interrupt exception processing. During the core's exception processing, the IACK cycle performs and the interrupt controller returns the appropriate vector number. As the interrupt acknowledge read performs, the vector number returns to the core. The contents of the CLMASK register load into the SLMASK register, and the CLMASK register updates to the level of the acknowledge interrupt. Additionally, the processor raises the interrupt mask in the status register (SR[I]) to match the level of the acknowledged request. At the end of the core's exception processing, control passes to the interrupt service routine (ISR), shown as the beginning of segment C.

During C, the initial portion of the ISR executes. Near the end of this segment, the ISR accesses the peripheral to negate the interrupt request source. At the conclusion of segment C, the SR[I] field can be lowered to re-enable interrupts with a priority greater than the original request.

The bulk of the interrupt service routine executes in segment D, with interrupts enabled. Near the end of the service routine, the SR[I] field is again raised to the original acknowledged level, preparing to perform the context switch.

At the end of segment E, the original value in the saved level mask (SLMASK) is restored in the current level mask (CLMASK). Optionally, the service routine can directly load the CLMASK register with any value with pending interrupt requests of certain levels need to be examined.

In segment F, the interrupt service routine completes execution. During this period of time, it is possible to access the interrupt controller with a software IACK to see if there are any pending properly-enabled requests. Checking for any pending interrupt requests at this time provides ability to initiate processing of another interrupt without the need to return from the original and incur the overhead of another interrupt exception.

At the conclusion of segment G, the processor core returns to the original interrupted task or a different task ready to execute.

Obviously, there are many variations to the managing of the SR[I] and the CLMASK values to create a flexible, responsive system for managing interrupt requests within the device.

Chapter 18
Edge Port Module (EPORT)

18.1 Introduction

The edge port module (EPORT) has up to eight interrupt pins, $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ0}}$. Each pin can be configured individually as a level-sensitive interrupt pin, an edge-detecting interrupt pin (rising edge, falling edge, or both), or a general-purpose input/output (I/O) pin.

NOTE

Not all EPORT signals may be output from the device. See Chapter 2, "Signal Descriptions," to determine which signals are available.

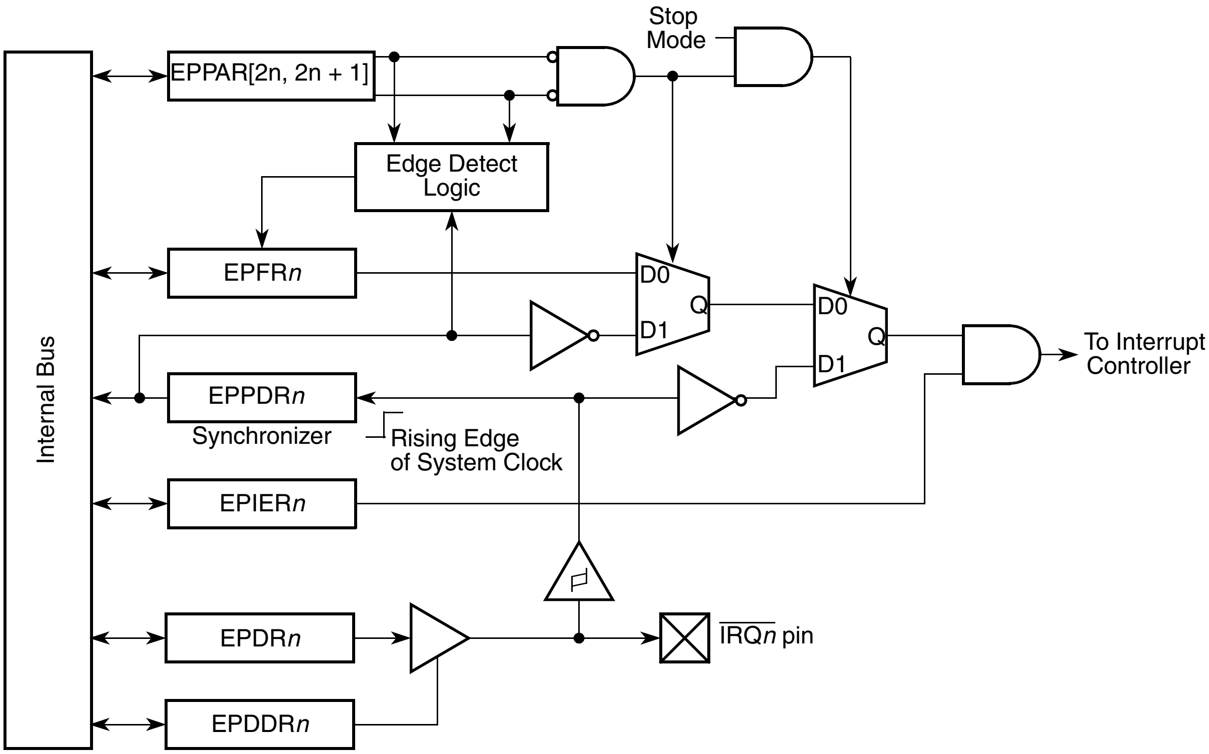


Figure 18-1. EPORT Block Diagram

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, "Pin Multiplexing and Control") prior to configuring the edge-port module.

18.2 Low-Power Mode Operation

This section describes the operation of the EPORT module in low-power modes. For more information on low-power modes, see Chapter 9, “Power Management”. Table 18-1 shows EPORT-module operation in low-power modes and describes how this module may exit each mode.

NOTE

The wakeup control register (WCR) in the system control module specifies the interrupt level at or above what is needed to bring the device out of a low-power mode.

Table 18-1. Edge Port Module Operation in Low-Power Modes

Low-power Mode	EPORT Operation	Mode Exit
Wait	Normal	Any $\overline{\text{IRQ}}n$ interrupt at or above level in WCR
Doze	Normal	Any $\overline{\text{IRQ}}n$ interrupt at or above level in WCR
Stop	Level-sensing only	Any $\overline{\text{IRQ}}n$ interrupt set for level-sensing at or above level in WCR. See note below.

In wait and doze modes, the EPORT module continues to operate as it does in run mode. It may be configured to exit the low-power modes by generating an interrupt request on a selected edge or a low level on an external pin. In stop mode, no clocks are available to perform the edge-detect function. Only the level-detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

NOTE

In stop mode, the input pin synchronizer is bypassed for the level-detect logic because no clocks are available.

18.3 Signal Descriptions

All EPORT pins default to general-purpose input pins at reset. The pin value is synchronized to the rising edge of FB_CLK when read from the EPORT pin data register (EPPDR). The values used in the edge/level detect logic are also synchronized to the rising edge of FB_CLK. These pins use Schmitt-triggered input buffers with built-in hysteresis designed to decrease the probability of generating false, edge-triggered interrupts for slow rising and falling input signals.

When a pin is configured as an output, it is driven to a state whose level is determined by the corresponding bit in the EPORT data register (EPDR). All bits in the EPDR are set at reset.

18.4 Memory Map/Register Definition

This subsection describes the memory map and register structure. Refer to Table 18-2 for a description of the EPORT memory map.

NOTE

Longword accesses to any of the edge-port registers result in a bus error. Only byte and word accesses are allowed.

Table 18-2. Edge Port Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers[1]					
0xFC09_4000	EPORT Pin Assignment Register (EPPAR)	16	R/W	0x0000	18.4.1/18-3
0xFC09_4002	EPORT Data Direction Register (EPDDR)	8	R/W	0x00	18.4.2/18-4
0xFC09_4003	EPORT Interrupt Enable Register (EPIER)	8	R/W	0x00	18.4.3/18-5
Supervisor/User Access Registers					
0xFC09_4004	EPORT Data Register (EPDR)	8	R/W	0xFF	18.4.4/18-5
0xFC09_4005	EPORT Pin Data Register (EPPDR)	8	R	See Section	18.4.5/18-5
0xFC09_4006	EPORT Flag Register (EPFR)	8	R/W	0x00	18.4.6/18-6

[1] User access to supervisor-only address locations have no effect and result in a bus error.

18.4.1 EPORT Pin Assignment Register (EPPAR)

The EPORT pin assignment register (EPPAR) controls the function of each pin individually.

Address: 0xFC09_4000 (EPPAR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	EPPA7		EPPA6		EPPA5		EPPA4		EPPA3		EPPA2		EPPA1		EPPA0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-2. EPORT Pin Assignment Register (EPPAR)

Table 18-3. EPPAR Field Descriptions

Field	Description
15–0 EPPA*n*	EPORT pin assignment select fields. The read/write EPPA*n* fields configure EPORT pins for level detection and rising and/or falling edge detection. Pins configured as level-sensitive are active-low (logic 0 on the external pin represents a valid interrupt request). Level-sensitive interrupt inputs are not latched. To guarantee that a level-sensitive interrupt request is acknowledged, the interrupt source must keep the signal asserted until acknowledged by software. Level sensitivity must be selected to bring the device out of stop mode with an $\overline{\mathrm{IRQ}n}$ interrupt. Pins configured as edge-triggered are latched and need not remain asserted for interrupt generation. A pin configured for edge detection can trigger an interrupt regardless of its configuration as input or output. Interrupt requests generated in the EPORT module can be masked by the interrupt controller module. EPPAR functionality is independent of the selected pin direction. Reset clears the EPPA*n* fields. 00 Pin $\overline{\mathrm{IRQ}n}$ level-sensitive 01 Pin $\overline{\mathrm{IRQ}n}$ rising edge triggered 10 Pin $\overline{\mathrm{IRQ}n}$ falling edge triggered 11 Pin $\overline{\mathrm{IRQ}n}$ falling edge and rising edge triggered

18.4.2 EPORT Data Direction Register (EPDDR)

The EPORT data direction register (EPDDR) controls the direction of each one of the pins individually.

Address: 0xFC09_4002 (EPDDR) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R W	EPDD7	EPDD6	EPDD5	EPDD4	EPDD3	EPDD2	EPDD1	EPDD0
Reset	0	0	0	0	0	0	0	0

Figure 18-3. EPORT Data Direction Register (EPDDR)

Table 18-4. EPDDR Field Descriptions

Field	Description
7–0 EPDD*n*	Setting any bit in the EPDDR configures the corresponding pin as an output. Clearing any bit in EPDDR configures the corresponding pin as an input. Pin direction is independent of the level/edge detection configuration. Reset clears EPDD7–EPDD0. To use an EPORT pin as an external interrupt request source, its corresponding bit in EPDDR must be clear. Software can generate interrupt requests by programming the EPORT data register when the EPDDR selects output. 0 Corresponding EPORT pin configured as input 1 Corresponding EPORT pin configured as output

18.4.3 Edge Port Interrupt Enable Register (EPIER)

The EPORT interrupt enable register (EPIER) enables interrupt requests for each pin individually.

Address: 0xFC09_4003 (EPIER) Access: User read/write

	7	6	5	4	3	2	1	0
R W	EPIE7	EPIE6	EPIE5	EPIE4	EPIE3	EPIE2	EPIE1	EPIE0
Reset:	0	0	0	0	0	0	0	0

Figure 18-4. EPORT Port Interrupt Enable Register (EPIER)

Table 18-5. EPIER Field Descriptions

Field	Description
7–0 EPIE*n*	Edge port interrupt enable bits enable EPORT interrupt requests. If a bit in EPIER is set, EPORT generates an interrupt request when: • The corresponding bit in the EPORT flag register (EPFR) is set or later becomes set • The corresponding pin level is low and the pin is configured for level-sensitive operation Clearing a bit in EPIER negates any interrupt request from the corresponding EPORT pin. Reset clears EPIE7–EPIE0. 0 Interrupt requests from corresponding EPORT pin disabled 1 Interrupt requests from corresponding EPORT pin enabled

18.4.4 Edge Port Data Register (EPDR)

The EPORT data register (EPDR) holds the data to be driven to the pins.

Address: 0xFC09_4004 (EPDR) Access: User read/write

	7	6	5	4	3	2	1	0
R W	EPD7	EPD6	EPD5	EPD4	EPD3	EPD2	EPD1	EPD0
Reset:	1	1	1	1	1	1	1	1

Figure 18-5. EPORT Port Data Register (EPDR)

Table 18-6. EPDR Field Descriptions

Field	Description
7–0 EPD*n*	Edge port data bits. An internal register stores data written to EPDR; if any pin of the port is configured as an output, the bit stored for that pin is driven onto the pin. Reading EDPR returns the data stored in the register. Reset sets EPD7 – EPD0.

18.4.5 Edge Port Pin Data Register (EPPDR)

The EPORT pin data register (EPPDR) reflects the current state of the pins.

Address: 0xFC09_4005 (EPPDR) Access: User read-only

	7	6	5	4	3	2	1	0
R	EPPD7	EPPD6	EPPD5	EPPD4	EPPD3	EPPD2	EPPD1	EPPD0
W								
Reset:	[$\overline{\text{IRQ7}}$]	[$\overline{\text{IRQ6}}$]	[$\overline{\text{IRQ5}}$]	[$\overline{\text{IRQ4}}$]	[$\overline{\text{IRQ3}}$]	[$\overline{\text{IRQ2}}$]	[$\overline{\text{IRQ1}}$]	[$\overline{\text{IRQ0}}$]

Figure 18-6. EPORT Port Pin Data Register (EPPDR)

Table 18-7. EPPDR Field Descriptions

Field	Description
7–0 EPPD*n*	Edge port pin data bits. The read-only EPPDR reflects the current state of the EPORT pins $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ0}}$. Writing to EPPDR has no effect, and the write cycle terminates normally. Reset does not affect EPPDR.

18.4.6 Edge Port Flag Register (EPFR)

The EPORT flag register (EPFR) individually latches EPORT edge events.

Address: 0xFC09_4006 (EPFR) Access: User read/write

	7	6	5	4	3	2	1	0
R	EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	EPF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 18-7. EPORT Port Flag Register (EPFR)

Table 18-8. EPFR Field Descriptions

Field	Description
7–0 EPF*n*	Edge port flag bits. When an EPORT pin is configured for edge triggering, its corresponding read/write bit in EPFR indicates that the selected edge has been detected. Reset clears EPF7 – EPF0. Bits in this register are set when the selected edge is detected on the corresponding pin. A bit remains set until cleared by writing a 1 to it. Writing 0 has no effect. If a pin is configured as level-sensitive (EPPAR*n* = 00), pin transitions do not affect this register. 0 Selected edge for $\overline{\text{IRQ}n}$ pin not detected 1 Selected edge for $\overline{\text{IRQ}n}$ pin detected

Chapter 19
Enhanced Direct Memory Access (eDMA)

19.1 Overview

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data transfers with minimal intervention from a host processor. The hardware microarchitecture includes a DMA engine that performs source- and destination-address calculations, and the actual data-movement operations, along with local memory containing transfer control descriptors for each channel.

19.1.1 Block Diagram

Figure 19-1 is a block diagram of the eDMA module.

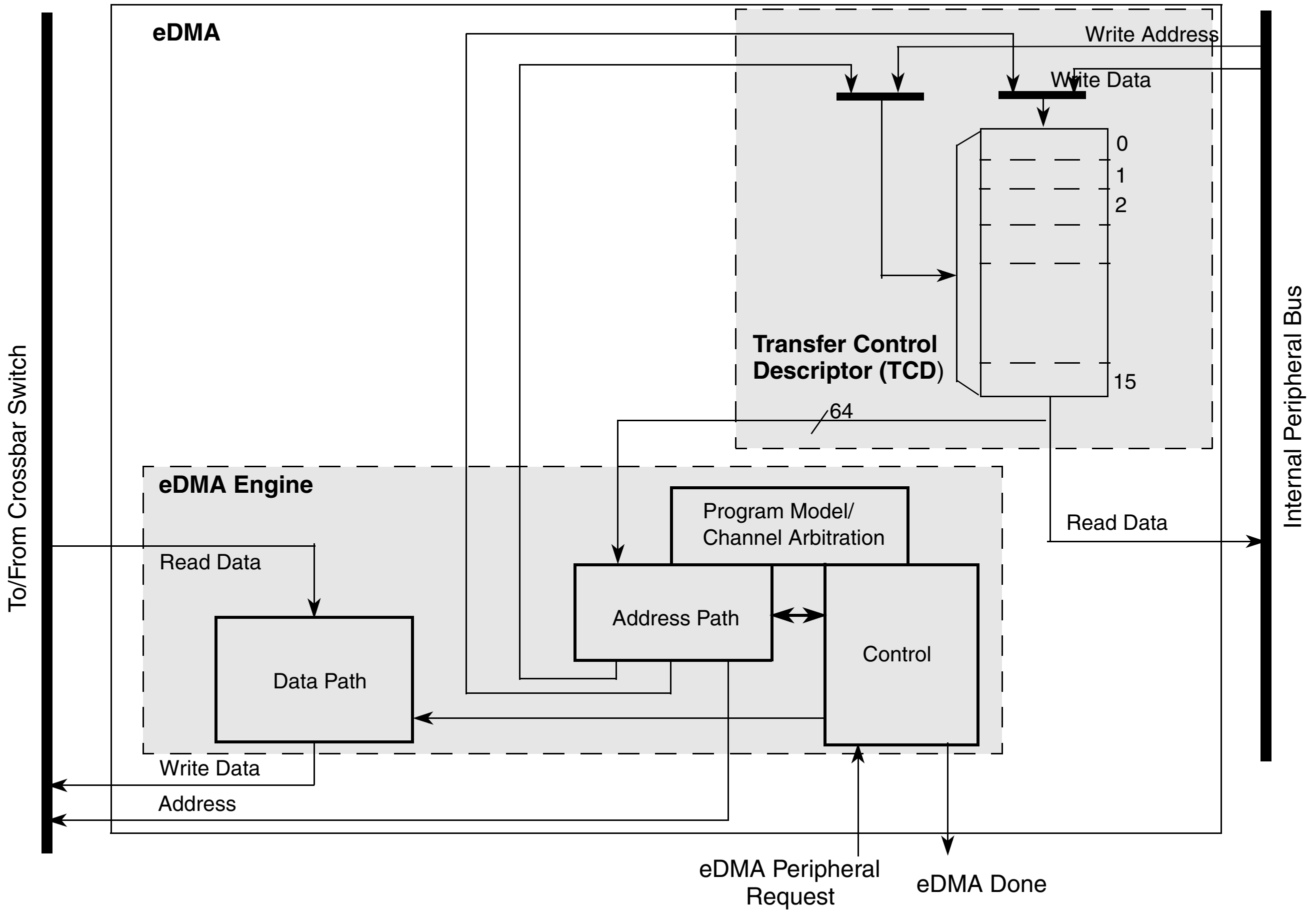


Figure 19-1. eDMA Block Diagram

19.1.2 Features

The eDMA is a highly-programmable data-transfer engine optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known and not defined within the data packet itself. The eDMA module features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source and destination addresses and transfer size, plus support for enhanced addressing modes
- 16-channel implementation that performs complex data transfers with minimal intervention from a host processor
  - Internal data buffer, used as temporary storage to support 16-byte burst transfers
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD stored in local memory for each channel
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continual transfers
  - Peripheral-paced hardware requests (one per channel)
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel and logically summed together to form one error interrupt to the interrupt controller
- Optional support for scatter/gather DMA processing

Throughout this chapter, *n* is used to reference the channel number.

19.2 Modes of Operation

19.2.1 Normal Mode

In normal mode, the eDMA transfers data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

A service request initiates a transfer of a specific number of bytes (NBYTES) as specified in the transfer control descriptor (TCD). The minor loop is the sequence of read-write operations that transfers these NBYTES per service request. A major loop is the number of minor loop iterations defining a task.

19.2.2 Debug Mode

In debug mode, the eDMA stops transferring data. If debug mode is entered during the transfer of a data block described by a minor loop in the current active channel's TCD, the eDMA continues operation until completion of the minor loop.

19.3 External Signal Description

This section describes the external signals of the eDMA controller.

Table 19-1. External Signal List

Signal Name	I/O	Description
$\overline{\text{DREQ0}}$ $\overline{\text{DREQ1}}$	I	Provides external requests from peripherals needing DMA service. When asserted, the device is requesting service. This request pin is tied to DMA channel 0 and 1, respectively.
$\overline{\text{DACK0}}$ $\overline{\text{DACK1}}$	O	Indicates when the external DMA request has been acknowledged.

19.3.1 External Signal Timing

Asserting the external DMA request signal, $\overline{\text{DREQ}n}$, initiates a service request for that channel. It must remain asserted until the corresponding $\overline{\text{DACK}n}$ signal indicates the channel's data transfer has started. The $\overline{\text{DACK}n}$ output is asserted for one cycle during the address phase of the channel's first internal read access.

- When no further requests are needed, the $\overline{\text{DREQ}n}$ signal must negate after the $\overline{\text{DACK}n}$ assertion and on or before the second cycle following the data phase of the last internal bus write (see Figure 19-2).
- If another service request is needed, $\overline{\text{DREQ}n}$ may simply remain asserted.
- To request continuous service, $\overline{\text{DREQ}n}$ may remain continuously asserted.

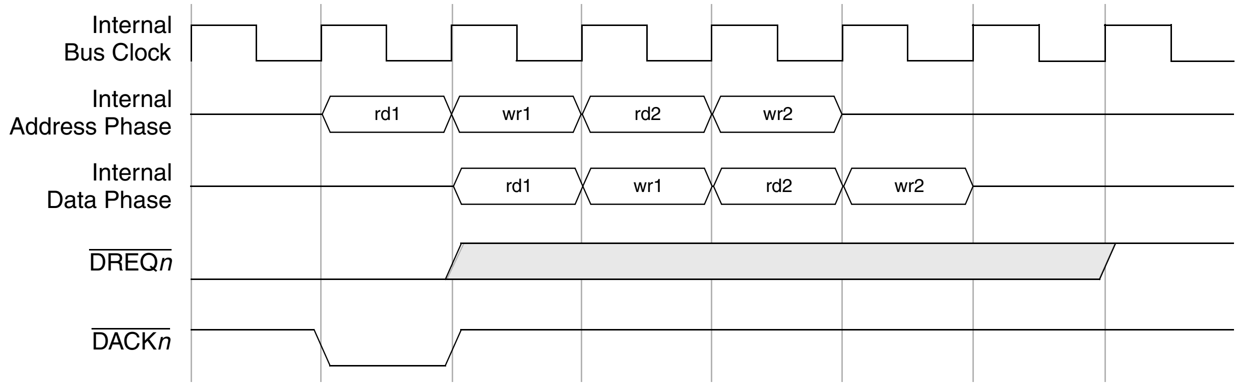


Figure 19-2. $\overline{\text{DREQ}n}$ and $\overline{\text{DACK}n}$ Timing

After a service request has been initiated, it cannot be canceled. Removing a service request after it has been asserted may result in one of three actions depending on the DMA engine's status:

- The request is never recognized because another channel is executing.

- The request is considered spurious and discarded, because the request is removed during arbitration for next channel selection.
- The channel is selected by arbitration and begins execution.

19.4 Memory Map/Register Definition

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions, while the second region corresponds to the local transfer control descriptor memory.

Reading reserved bits in a register return the value of zero and writes to reserved bits in a register are ignored. Reading or writing to a reserved memory location generates a bus error.

Table 19-2. eDMA Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC04_4000	eDMA Control Register (EDMA_CR)	32	R/W	0x0000_0000	19.4.1/19-4
0xFC04_4004	eDMA Error Status Register (EDMA_ES)	32	R	0x0000_0000	19.4.2/19-5
0xFC04_400E	eDMA Enable Request Register (EDMA_ERQ)	16	R/W	0x0000	19.4.3/19-8
0xFC04_4016	eDMA Enable Error Interrupt Register (EDMA_EEI)	16	R/W	0x0000	19.4.4/19-9
0xFC04_4018	eDMA Set Enable Request (EDMA_SERQ)	8	W	0x00	19.4.5/19-10
0xFC04_4019	eDMA Clear Enable Request (EDMA_CERQ)	8	W	0x00	19.4.6/19-10
0xFC04_401A	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	8	W	0x00	19.4.7/19-11
0xFC04_401B	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	8	W	0x00	19.4.8/19-11
0xFC04_401C	eDMA Clear Interrupt Request Register (EDMA_CINT)	8	W	0x00	19.4.9/19-12
0xFC04_401D	eDMA Clear Error Register (EDMA_CERR)	8	W	0x00	19.4.10/19-13
0xFC04_401E	eDMA Set START Bit Register (EDMA_SSRT)	8	W	0x00	19.4.11/19-13
0xFC04_401F	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	8	W	0x00	19.4.12/19-14
0xFC04_4026	eDMA Interrupt Request Register (EDMA_INT)	32	R/W	0x0000	19.4.13/19-15
0xFC04_402E	eDMA Error Register (EDMA_ERR)	32	R/W	0x0000	19.4.14/19-15
0xFC04_4100 + hex(*n*)	eDMA Channel *n* Priority Register (DCHPRI*n*) for *n* = 0 – 15	8	R/W	See Section	19.4.15/19-16
0xFC04_5000 + hex(32×*n*)	Transfer Control Descriptor (TCD*n*) for *n* = 0 – 15	256	R/W	See Section	19.4.16/19-17

19.4.1 eDMA Control Register (EDMA_CR)

The EDMA_CR defines the basic operating configuration of the eDMA. Arbitration can be configured to use a fixed-priority or a round-robin scheme. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The channel priority registers assign the priorities (see Section 19.4.15, "eDMA Channel n Priority Registers (DCHPRIn)"). In round-robin arbitration mode, the channel priorities are ignored, and channels are cycled through without regard to priority.

NOTE

For proper operation, writes to the EDMA_CR register must only be performed when the DMA channels are inactive (TCR*n*_CSR[ACTIVE] bits are cleared).

Address: 0xFC04_4000 (EDMA_CR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-3. eDMA Control Register (EDMA_CR)

Table 19-3. EDMA_CR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–3	Reserved, should be cleared.
2 ERCA	Enable round robin channel arbitration. 0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
1 EDBG	Enable debug. 0 When in debug mode the DMA continues to operate. 1 When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when the system exits debug mode or the EDBG bit is cleared.
0	Reserved, must be cleared.

19.4.2 eDMA Error Status Register (EDMA_ES)

The EDMA_ES provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer-control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is reported when the starting source or destination address, source or destination offsets, minor loop byte count, or the transfer size represent an inconsistent state. Each of these possible causes are detailed in the below list:

- The addresses and offsets must be aligned on 0-modulo-transfer-size boundaries
- The minor loop byte count must be a multiple of the source and destination transfer sizes.
- All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

- In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal. All channel priority levels must be unique when fixed arbitration mode is enabled.
- If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (DLAST_SGA) is not aligned on a 32-byte boundary.
- If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD*n*_CITER[E_LINK] bit does not equal the TCD*n*_BITER[E_LINK] bit.

If enabled, all configuration error conditions, except the scatter/gather and minor-loop link errors, report as the channel activates and asserts an error interrupt request. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system-bus error occurs, the channel terminates after the read or write transaction (which is already pipelined after errant access) has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel terminates due to the destination bus error.

The occurrence of any error causes the eDMA engine to stop the active channel immediately, and the appropriate channel bit in the eDMA error register is asserted. At the same time, the details of the error condition are loaded into the EDMA_ES. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the eDMA engine continues operating by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel executes and terminate with the same error condition.

Address: 0xFC04_4004 (EDMA_ES) Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	CPE	0	0	ERRCHN				SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-4. eDMA Error Status Register (EDMA_ES)

Table 19-4. EDMA_ES Field Descriptions

Field	Description
31 VLD	Logical OR of all EDMA_ERR status bits 0 No EDMA_ERR bits are set 1 At least one EDMA_ERR bit is set indicating a valid error exists that has not been cleared
30–15	Reserved, must be cleared.
14 CPE	Channel priority error 0 No channel priority error 1 The last recorded error was a configuration error in the channel priorities. Channel priorities within are not unique.
13–12	Reserved, must be cleared.
11–8 ERRCHN	Error channel number. The channel number of the last recorded error (excluding CPE errors).
7 SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_SADDR field. TCD*n*_SADDR is inconsistent with TCD*n*_ATTR[SSIZE]
6 SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_SOFF field. TCD*n*_SOFF is inconsistent with TCD*n*_ATTR[SSIZE].
5 DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_DADDR field. TCD*n*_DADDR is inconsistent with TCD*n*_ATTR[DSIZE].
4 DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_DOFF field. TCD*n*_DOFF is inconsistent with TCD*n*_ATTR[DSIZE].
3 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_NBYTES or TCD*n*_CITER fields. • TCD*n*_NBYTES is not a multiple of TCD*n*_ATTR[SSIZE] and TCD*n*_ATTR[DSIZE], or • TCD*n*_CITER[CITER] is equal to zero, or • TCD*n*_CITER[E_LINK] is not equal to TCD*n*_BITER[E_LINK].
2 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_DLAST_SGA field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD*n*_CSR[E_SG] is enabled. TCD*n*_DLAST_SGA is not on a 32 byte boundary.
1 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
0 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

19.4.3 eDMA Enable Request Register (EDMA_ERQ)

The EDMA_ERQ register provides a bit map for the 16 implemented channels to enable the request signal for each channel. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the EDMA_SERQ and EDMA_CERQ. The EDMA_{S,C}ERQR are provided so the request enable for a single channel can easily be modified without needing to perform a read-modify-write sequence to the EDMA_ERQ.

DMA request input signals and this enable request flag must be asserted before a channel’s hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made explicitly through software or a linked channel request.

Address: 0xFC04_400E (EDMA_ERQ) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 9	ERQ 8	ERQ 7	ERQ 6	ERQ 5	ERQ 4	ERQ 3	ERQ 2	ERQ 1	ERQ 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-5. eDMA Enable Request Register (EDMA_ERQ)

Table 19-5. EDMA_ERQ Field Descriptions

Field	Description
15–0 ERQ*n*	Enable DMA Request *n.* 0 The DMA request signal for channel *n* is disabled. 1 The DMA request signal for channel *n* is enabled.

The assignments between the DMA requests from the peripherals to the channels of the eDMA are shown in Table 19-6.

Table 19-6. DMA Request Summary for eDMA

Channel	Source	Description
0	$\overline{\text{DREQ0}}$	External DMA request 0
1	$\overline{\text{DREQ1}}$	External DMA request 1
2	UISR0[FFULL/RXRDY]	UART0 Receive
3	UISR0[TXRDY]	UART0 Transmit
4	UISR1[FFULL/RXRDY]	UART1 Receive
5	UISR1[TXRDY]	UART1 Transmit
6	UISR2[FFULL/RXRDY]	UART2 Receive
7	UISR2[TXRDY]	UART2 Transmit
8	DTER0[CAP] or DTER0[REF] / SSISR[RFF0]	Timer 0 / SSI0 Receive[1]
9	DTER1[CAP] or DTER1[REF] / SSISR[RFF1]	Timer 1 / SSI1 Receive[1]

Table 19-6. DMA Request Summary for eDMA (continued)

Channel	Source	Description
10	DTER2[CAP] or DTER2[REF] / SSISR[TFE0]	Timer 2 / SSI0 Transmit[1]
11	DTER3[CAP] or DTER3[REF] / SSISR[TFE1]	Timer 3 / SSI1 Transmit[1]
12	DSPI_SR[RFDF]	DSPI Receive
13	DSPI_SR[TFFF]	DSPI Transmit
14	ATA_ISR[DMA]	ATA Receive
15	ATA_ISR[DMA]	ATA Transmit

[1] For information on how to select between SSI and Timer sources, refer to Chapter 11, “Chip Configuration Module (CCM).”

As a given channel completes the processing of its major iteration count, a flag in the transfer control descriptor that affect the ending state of the EDMA_ERQ bit for that channel. If the TCD*n*_CSR[D_REQ] bit is set, the corresponding EDMA_ERQ bit is cleared, disabling the DMA request. If the D_REQ bit clears, the state of the EDMA_ERQ bit is unaffected.

19.4.4 eDMA Enable Error Interrupt Registers (EDMA_EEI)

The EDMA_EEI register provides a bit map for the 16 channels to enable the error interrupt signal for each channel. The state of any given channel’s error interrupt enable is directly affected by writes to this register; it is also affected by writes to the EDMA_SEEI and EDMA_CEEI. The EDMA_{S,C}EEIR are provided so the error interrupt enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_EEI register.

The DMA error indicator and the error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted to the interrupt controller.

Address: 0xFC04_4016 (EDNA_EEI) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI9	EEI8	EEI7	EEI6	EEI5	EEI4	EEI3	EEI2	EEI1	EEI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-6. eDMA Enable Error Interrupt Register (EDMA_EEI)

Table 19-7. EDMA_EEI Field Descriptions

Field	Description
15–0 EEI*n*	Enable error interrupt *n.* 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generates an error interrupt request.

19.4.5 eDMA Set Enable Request Register (EDMA_SERQ)

The EDMA_SERQ provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQ to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQ to be set. Setting the SAER bit provides a global set function, forcing the entire contents of EDMA_ERQ to be set. Reads of this register return all zeroes.

Address: 0xFC04_4018 (EDMA_SERQ) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAER			SERQ			
Reset	0	0	0	0	0	0	0	0

Figure 19-7. eDMA Set Enable Request Register (EDMA_SERQ)

Table 19-8. EDMA_SERQ Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SAER	Set all enable requests. 0 Set only those EDMA_ERQ bits specified in the SERQ field. 1 Set all bits in EDMA_ERQ.
5–4	Reserved, must be cleared.
3–0 SERQ	Set enable request. Sets the corresponding bit in EDMA_ERQ

19.4.6 eDMA Clear Enable Request Register (EDMA_CERQ)

The EDMA_CERQ provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQ to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQ to be cleared. Setting the CAER bit provides a global clear function, forcing the entire contents of the EDMA_ERQ to be cleared, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: 0xFC04_4019 (EDMA_CERQ) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAER			CERQ			
Reset	0	0	0	0	0	0	0	0

Figure 19-8. eDMA Clear Enable Request Register (EDMA_CERQ)

Table 19-9. EDMA_CERQ Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAER	Clear all enable requests. 0 Clear only those EDMA_ERQ bits specified in the CERQ field. 1 Clear all bits in EDMA_ERQ.
5–4	Reserved, must be cleared.
3–0 CERQ	Clear enable request. Clears the corresponding bit in EDMA_ERQ.

19.4.7 eDMA Set Enable Error Interrupt Register (EDMA_SEEI)

The EDMA_SEEI provides a simple memory-mapped mechanism to set a given bit in the EDMA_EEI to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEI to be set. Setting the SAEE bit provides a global set function, forcing the entire EDMA_EEI contents to be set. Reads of this register return all zeroes.

Address: 0xFC04_401A (EDMA_SEEI) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAEE			SEEI			
Reset	0	0	0	0	0	0	0	0

Figure 19-9. eDMA Set Enable Error Interrupt Register (EDMA_SEEI)

Table 19-10. EDMA_SEEI Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SAEE	Sets all enable error interrupts. 0 Set only those EDMA_EEI bits specified in the SEEI field. 1 Sets all bits in EDMA_EEI.
5–4	Reserved, must be cleared.
3–0 SEEI	Set enable error interrupt. Sets the corresponding bit in EDMA_EEI.

19.4.8 eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)

The EDMA_CEEI provides a simple memory-mapped mechanism to clear a given bit in the EDMA_EEI to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEI to be cleared. Setting the CAEE bit provides a global clear function, forcing the EDMA_EEI contents to be cleared, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: 0xFC04_401B (EDNA_CEEI)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAEE			CEEI			
Reset	0	0	0	0	0	0	0	0

Figure 19-10. eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)

Table 19-11. EDMA_CEEI Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAEE	Clear all enable error interrupts. 0 Clear only those EDMA_EEI bits specified in the CEEI field. 1 Clear all bits in EDMA_EEI.
5–4	Reserved, must be cleared.
3–0 CEEI	Clear enable error interrupt. Clears the corresponding bit in EDMA_EEI.

19.4.9 eDMA Clear Interrupt Request Register (EDMA_CINT)

The EDMA_CINT provides a simple, memory-mapped mechanism to clear a given bit in the EDMA_INT to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_INT to be cleared. Setting the CAIR bit provides a global clear function, forcing the entire contents of the EDMA_INT to be cleared, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Address: 0xFC04_401C (EDMA_CINT)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAIR			CINT			
Reset	0	0	0	0	0	0	0	0

Figure 19-11. eDMA Clear Interrupt Request (EDMA_CINT)

Table 19-12. EDMA_CINT Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAIR	Clear all interrupt requests. 0 Clear only those EDMA_INT bits specified in the CINT field. 1 Clear all bits in EDMA_INT.

Table 19-12. EDMA_CINT Field Descriptions (continued)

Field	Description
5–4	Reserved, must be cleared.
3–0 CINT	Clear interrupt request. Clears the corresponding bit in EDMA_INT.

19.4.10 eDMA Clear Error Register (EDMA_CERR)

The EDMA_CERR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERR to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERR to be cleared. Setting the CAEI bit provides a global clear function, forcing the EDMA_ERR contents to be cleared, clearing all channel error indicators. Reads of this register return all zeroes.

Address: 0xFC04_401D (EDMA_CERR) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAEI			CERR			
Reset	0	0	0	0	0	0	0	0

Figure 19-12. eDMA Clear Error Register (EDMA_CERR)

Table 19-13. EDMA_CERR Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAEI	Clear all error indicators. 0 Clear only those EDMA_ERR bits specified in the CERR field. 1 Clear all bits in EDMA_ERR.
5–4	Reserved, must be cleared.
3–0 CERR	Clear error indicator. Clears the corresponding bit in EDMA_ERR.

19.4.11 eDMA Set START Bit Register (EDMA_SSRT)

The EDMA_SSRT provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting the SAST bit provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Address: 0xFC04_401E (EDMA_SSRT) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAST			SSRT			
Reset	0	0	0	0	0	0	0	0

Figure 19-13. eDMA Set START Bit Register (EDMA_SSRT)

Table 19-14. EDMA_SSRT Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SAST	Set all START bits (activates all channels). 0 Set only those TCD*n*_CSR[START] bits specified in the SSRT field. 1 Set all bits in TCD*n*_CSR[START].
5–4	Reserved, must be cleared.
3–0 SSRT	Set START bit. Sets the corresponding bit in TCD*n*_CSR[START].

19.4.12 eDMA Clear DONE Status Bit Register (EDMA_CDNE)

The EDMA_CDNE provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting the CADN bit provides a global clear function, forcing all DONE bits to be cleared. Reads of this register return all zeroes.

Address: 0xFC04_401F (EDMA_CDNE) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CADN			CDNE			
Reset	0	0	0	0	0	0	0	0

Figure 19-14. eDMA Clear DONE Status Bit Register (EDMA_CDNE)

Table 19-15. EDMA_CDNE Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CADN	Clears all DONE bits. 0 Clears only those TCD*n*_CSR[DONE] bits specified in the CDNE field. 1 Clears all bits in TCD*n*_CSR[DONE]
5–4	Reserved, must be cleared.
3–0 CDNE	Clear DONE bit. Clears the corresponding bit in TCD*n*_CSR[DONE].

19.4.13 eDMA Interrupt Request Register (EDMA_INT)

The EDMA_INT provide a bit map for the 16 channels signaling the presence of an interrupt request for each channel. Depending on the appropriate bit setting in the transfer-control descriptions, the eDMA engine generates an interrupt a data transfer completion. The outputs of this register are directly routed to the interrupt controller (INTC). During the interrupt-service routine associated with any given channel, it is the software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CINT in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CINT. On writes to the EDMA_INT, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CINT is provided so the interrupt request for a single channel can easily be cleared without the need to perform a read-modify-write sequence to the EDMA_INT.

Address: 0xFC04_4026 (EDMA_INT) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INT15	INT14	INT13	INT12	INT11	INT10	INT9	INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-15. eDMA Interrupt Request Register (EDMA_INT)

Table 19-16. EDMA_INT Field Descriptions

Field	Description
15–0 INT*n*	eDMA interrupt request *n* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

19.4.14 eDMA Error Register (EDMA_ERR)

The EDMA_ERR provide a bit map for the 16 channels, signaling the presence of an error for each channel. The eDMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEI,and then routed to the interrupt controller. During the execution of the interrupt-service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error-interrupt request. Typically, a write to the EDMA_CERR in the interrupt-service routine is used for this purpose. The normal DMA channel completion indicators (setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request) are not affected when an error is detected.

The contents of this register can also be polled because a non-zero value indicates the presence of a channel error regardless of the state of the EDMA_EEI. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CERR. On writes to the EDMA_ERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The EDMA_CERR is provided so the error indicator for a single channel can easily be cleared.

Address: 0xFC04_402E (EDMA_ERR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 9	ERR 8	ERR 7	ERR 6	ERR 5	ERR 4	ERR 3	ERR 2	ERR 1	ERR 0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-16. eDMA Error Register (EDMA_ERR)

Table 19-17. EDMA_ERR Field Descriptions

Field	Description
15–0 ERR*n*	eDMA Error *n*. 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

19.4.15 eDMA Channel *n* Priority Registers (DCHPRI*n*)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; for example, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. Software must program the channel priorities with unique values. Otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per-channel basis by setting the DCHPRI*n*[ECP] bit. Channel preemption allows the executing channel's data transfers to temporarily suspend in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected.

Address: 0xFC04_4100 + *n*, where *n* = 0 – 15 (DCHPRI*n*) Access: User read/write

	7	6	5	4	3	2	1	0
R	ECP	0	0	0	CHPRI			
W								
Reset	0	0	0	0	–[1]	–[1]	–[1]	–[1]

[1] Reset value for the channel priority fields, CHPRI, is equal to the corresponding channel number for each priority register, i.e., DCHPRI15[CHPRI] equals 0b1111.

Figure 19-17. eDMA Channel *n* Priority Register (DCHPRI*n*)

Table 19-18. DCHPRI*n* Field Descriptions

Field	Description
7 ECP	Enable channel preemption. 0 Channel *n* cannot be suspended by a higher priority channel’s service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
6–4	Reserved, must be cleared.
3–0 CHPRI	Channel *n* arbitration priority. Channel priority when fixed-priority arbitration is enabled.

19.4.16 Transfer Control Descriptors (TCD*n*)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. Each TCD*n* definition is presented as 11 registers of 16 or 32 bits. Table 19-19 is a register list of the basic TCD structure.

Table 19-19. TCD*n* Memory Structure

eDMA Offset	TCD*n* Register Name	Abbreviation	Width (bits)
0xFC04_5000 + (0x20 × *n*)	Source Address	TCD*n*_SADDR	32
0xFC04_5004 + (0x20 × *n*)	Transfer Attributes	TCD*n*_ATTR	16
0xFC04_5006 + (0x20 × *n*)	Signed Source Address Offset	TCD*n*_SOFF	16
0xFC04_5008 + (0x20 × *n*)	Minor Byte Count	TCD*n*_NBYTES	32
0xFC04_500C + (0x20 × *n*)	Last Source Address Adjustment	TCD*n*_SLAST	32
0xFC04_5010 + (0x20 × *n*)	Destination Address	TCD*n*_DADDR	32
0xFC04_5014 + (0x20 × *n*)	Current Minor Loop Link, Major Loop Count	TCD*n*_CITER	16
0xFC04_5016 + (0x20 × *n*)	Signed Destination Address Offset	TCD*n*_DOFF	16
0xFC04_5018 + (0x20 × *n*)	Last Destination Address Adjustment/Scatter Gather Address	TCD*n*_DLAST_SGA	32
0xFC04_501C + (0x20 × *n*)	Beginning Minor Loop Link, Major Loop Count	TCD*n*_BITER	16
0xFC04_501E + (0x20 × *n*)	Control and Status	TCD*n*_CSR	16

The following figures and tables define the fields of the TCD*n* structure:

Address: 0xFC04_5000 + (0x20 × *n*) (TCD*n*_SADDR) Access: User 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
R W	SADDR																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 19-18. TCD*n* Source Address (TCD*n*_SADDR)

Table 19-20. TCD*n*_SADDR Field Descriptions

Field	Description
31–0 SADDR	Source address. Memory address pointing to the source data.

Address: 0xFC04_5004 + (0x20 × *n*) (TCD*n*_ATTR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SMOD					SSIZE			DMOD					DSIZE		
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 19-19. TCD*n* Transfer Attributes (TCD*n*_ATTR)

Table 19-21. TCD*n*_ATTR Field Descriptions

Field	Description
15–11 SMOD	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to implement a circular data queue easily. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits allowed to change. For a circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
10–8 SSIZE	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 100 16-byte Else Reserved The attempted use of a Reserved encoding causes a configuration error.
7–3 DMOD	Destination address modulo. See the SMOD definition.
2–0 DSIZE	Destination data transfer size. See the SSIZE definition.

Address: 0xFC04_5006 + (0x20 × *n*) (TCD*n*_SOFF) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SOFF															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 19-20. TCDn Signed Source Address Offset (TCD*n*_SOFF)

Table 19-22. TCD*n*_SOFF Field Descriptions

Field	Description
15–0 SOFF	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.

Address: 0xFC04_5008 + (0x20 × *n*) (TCD*n*_NBYTES) Access: User read/write

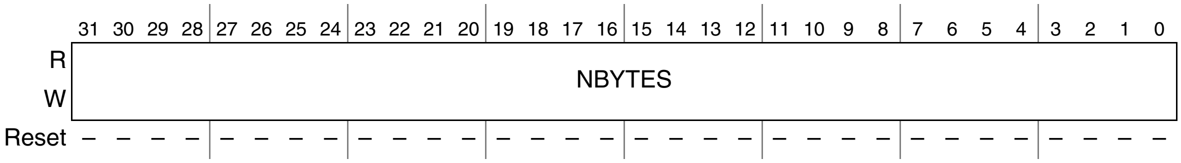


Figure 19-21. TCD*n* Minor Byte Count (TCD*n*_NBYTES)

Table 19-23. TCD*n*_NBYTES Field Descriptions

Field	Description
31–0 NBYTES	Minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel activates, the appropriate TCD contents load into the eDMA engine, and the appropriate reads and writes perform until the minor byte transfer count has transferred. This is an indivisible operation and cannot be halted. (Although, it may be stalled by using the bandwidth control field, or via preemption.) After the minor count is exhausted, the SADDR and DADDR values are written back into the TCD memory, the major iteration count is decremented and restored to the TCD memory. If the major iteration count is completed, additional processing is performed. **Note:** An NBYTES value of 0x0000_0000 is interpreted as a 4 GB transfer.

Address: 0xFC04_500C + (0x20 × *n*) (TCD*n*_SLAST) Access: User read/write

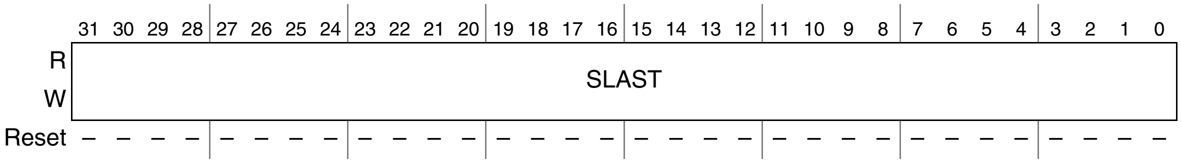


Figure 19-22. TCD*n* Source Last Address Adjustment (TCD*n*_SLAST)

Table 19-24. TCD*n*_SLAST Field Descriptions

Field	Description
31–0 SLAST	Last source address adjustment. Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to restore the source address to the initial value, or adjust the address to reference the next data structure.

Address: 0xFC04_5010 + (0x20 × *n*) (TCD*n*_DADDR) Access: User read/write

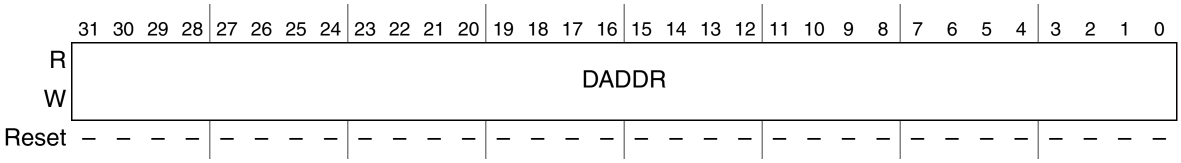


Figure 19-23. TCD*n* Destination Address (TCD*n*_DADDR)

Table 19-25. TCD*n*_DADDR Field Descriptions

Field	Description
31–0 DADDR	Destination address. Memory address pointing to the destination data.

Address: 0xFC04_5014 + (0x20 × *n*) (TCD*n*_CITER) Access: User read/write

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E_LINK = 1	R	E_LINK	0	0	LINKCH				CITER								
	W																
E_LINK = 0	R	E_LINK	CITER														
	W																
Reset		—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 19-24. TCD*n* Current Major Iteration Count (TCD*n*_CITER)

Table 19-26. TCD*n*_CITER Field Descriptions

Field	Description
15 E_LINK	Enable channel-to-channel linking on minor-loop complete. As the channel completes the minor loop, this flag enables linking to another channel, defined by the LINKCH field. The link target channel initiates a channel service request via an internal mechanism that sets the TCD*n*_CSR[START] bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR_E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER.E_LINK bit. Otherwise, a configuration error is reported.
14–13	Reserved, must be cleared.
12–9 LINKCH	Link channel number. If channel-to-channel linking is enabled (E_LINK = 1), then after the minor loop is exhausted, the eDMA engine initiates a channel service request to the channel defined by these four bits by setting that channel's TCD*n*_CSR[START] bit. 0–15 Link to DMA channel 0–15
14–0 or 8–0 CITER	Current major iteration count. This 9-bit (E_LINK = 1) or 15-bit (E_LINK = 0) count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (e.g., final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. **Note:** When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.

Address: 0xFC04_5016 + (0x20 × *n*) (TCD*n*_DOFF) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DOFF															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 19-25. TCD*n* Destination Address Signed Offset (TCD*n*_DOFF)

Table 19-27. TCD*n*_DOFF Field Descriptions

Field	Description
15–0 DOFF	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Address: 0xFC04_5018 + (0x20 × *n*) (TCD*n*_DLAST_SGA) Access: User 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
R W	DLAST_SGA																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 19-26. TCD*n* Destination Last Address Adjustment (TCD*n*_DLAST_SGA)

Table 19-28. TCD*n*_DLAST_SGA Field Descriptions

Field	Description
31–0 DLAST_SGA	Destination last address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If (TCD*n*_CSR[E_SG] = 0) then • Adjustment value added to the destination address at the completion of the major iteration count. This value can apply to restore the destination address to the initial value or adjust the address to reference the next data structure. else • This address points to the beginning of a 0-modulo-32-byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte, else a configuration error is reported.

Address: 0xFC04_501C + (0x20 × *n*) (TCD*n*_BITER) Access: User read/write

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E_LINK = 1	R	E_LINK	0	0	LINKCH				BITER								
	W		▨	▨													
E_LINK = 0	R W	E_LINK	BITER														
	Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 19-27. TCD*n* Beginning Major Iteration Count (TCD*n*_BITER)

Table 19-29. TCD*n*_BITER Field Descriptions

Field	Description
15 E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH. The link target channel initiates a channel service request via an internal mechanism that sets the TCD*n*_CSR[START] bit of the specified channel. If channel linking disables, the BITER value extends to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR_E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** When the software loads the TCD, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
14–13	Reserved, must be cleared.
12–9 LINKCH	Link channel number. If channel-to-channel linking is enabled (E_LINK = 1), then after the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by these four bits by setting that channel’s TCD*n*_CSR[START] bit. 0–15 Link to DMA channel 0–15 **Note:** When the software loads the TCD, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
14–0 or 8–0 BITER	Starting major iteration count. As the transfer control descriptor is first loaded by software, this 9-bit (E_LINK = 1) or 15-bit (E_LINK = 0) field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field. **Note:** When the software loads the TCD, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field. If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.

Address: 0xFC04_501E + (0x20 × *n*) (TCD*n*_CSR)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BWC		0	0	MAJOR_LINKCH				DONE	ACTIVE	MAJOR_E_LINK	E_SG	D_REQ	INT_HALF	INT_MAJOR	START
W																
Reset	—	—	—	—	—	—	—	—	0	0	—	—	—	—	—	0

Figure 19-28. TCD*n* Control and Status (TCD*n*_CSR)

Table 19-30. TCD*n*_CSR Field Descriptions

Field	Description
15–14 BWC	Bandwidth control. Throttles the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch (XBS). 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for 4 cycles after each r/w 11 eDMA engine stalls for 8 cycles after each r/w **Note:** If the source and destination sizes are equal, this field is ignored between the first and second transfers and after the last write of each minor loop. This behavior is a side effect of reducing start-up latency.
13–12	Reserved, must be cleared.
11–8 MAJOR_LINKCH	Link channel number. If (MAJOR_E_LINK = 0) then • No channel-to-channel linking (or chaining) is performed after the major loop counter is exhausted. else • After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these four bits by setting that channel's TCD*n*_CSR[START] bit. 0–15 Link to DMA channel 0–15
7 DONE	Channel done. This flag indicates the eDMA has completed the major loop. The eDMA engine sets it as the CITER count reaches zero; The software clears it, or the hardware when the channel is activated. **Note:** This bit must be cleared to write the MAJOR_E_LINK or E_SG bits.
6 ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and the eDMA clears it as the minor loop completes or if any error condition is detected.
5 MAJOR_E_LINK	Enable channel-to-channel linking on major loop complete. As the channel completes the major loop, this flag enables the linking to another channel, defined by MAJOR_LINKCH. The link target channel initiates a channel service request via an internal mechanism that sets the TCD*n*_CSR[START] bit of the specified channel. **Note:** To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD*n*_CSR[DONE] bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
4 E_SG	Enable scatter/gather processing. As the channel completes the major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure loaded as the transfer control descriptor into the local memory. **Note:** To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD*n*_CSR[DONE] bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the major loop completes its execution.
3 D_REQ	Disable request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQ bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQ bit is not affected. 1 The channel's EDMA_ERQ bit is cleared when the major loop is complete.

Table 19-30. TCD*n*_CSR Field Descriptions (continued)

Field	Description
2 INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_INT when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes or other types of data movement where the processor needs an early indication of the transfer's progress. The halfway complete interrupt disables when BITER values are less than two. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
1 INT_MAJOR	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_INT when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
0 START	Channel start. If this flag is set, the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

19.5 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

19.5.1 eDMA Microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer-control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules:

- eDMA Engine
  - Address Path:

    This block implements registered versions of two channel transfer control descriptors, channel x and channel y, and manages all master bus-address calculations. All the channels provide the same functionality. This structure allows data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel activation is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed, unless preempted by a higher priority channel. This provides a mechanism (enabled by DCHPRI*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

    When any channel is selected to execute, the contents of its TCD are read from local memory and loaded into the address path channel x registers for a normal start and into channel y registers for a preemption start. After the minor loop completes execution, the address path hardware writes the new values for the TCD*n*_{SADDR, DADDR, CITER} back to local memory. If the major iteration count is exhausted, additional processing are performed, including the final address pointer updates, reloading the TCD*n*_CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter/gather operation.

- Data Path:

  This block implements the bus master read/write datapath. It includes 16 bytes of register storage and the necessary multiplex logic to support any required data alignment. The internal read data bus is the primary input, and the internal write data bus is the primary output.

  The address and data path modules directly support the 2-stage pipelined internal bus. The address path module represents the 1st stage of the bus pipeline (address phase), while the data path module implements the 2nd stage of the pipeline (data phase).

- Program Model/Channel Arbitration:

  This block implements the first section of the eDMA programming model as well as the channel arbitration logic. The programming model registers are connected to the internal peripheral bus (not shown). The eDMA peripheral request inputs and interrupt request outputs are also connected to this block (via control logic).

- Control:

  This block provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read/destination write operations until the number of bytes specified in the minor loop byte count has moved. For descriptors where the sizes are not equal, multiple accesses of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- Transfer Control Descriptor Memory
  - Memory Controller:

    This logic implements the required dual-ported controller, managing accesses from the eDMA engine as well as references from the internal peripheral bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the peripheral transaction is stalled.

  - Memory Array: TCD storage is implemented using a single-port, synchronous RAM array.

19.5.2 eDMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 19-29, the first segment involves the channel activation. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel activation via software and the TCD*n*_CSR[START] bit follows the same basic flow as peripheral requests. The eDMA request input signal is registered internally and then routed through the eDMA engine: first through the control module, then into the program model and channel arbitration. In the next cycle, the channel arbitration performs, using the fixed-priority or round-robin algorithm. After arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the local memory for TCD*n*. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the eDMA engine address path channel x or y registers. The TCD memory is 64 bits wide to minimize the time needed to fetch the activated channel descriptor and load it into the address path channel x or y registers.

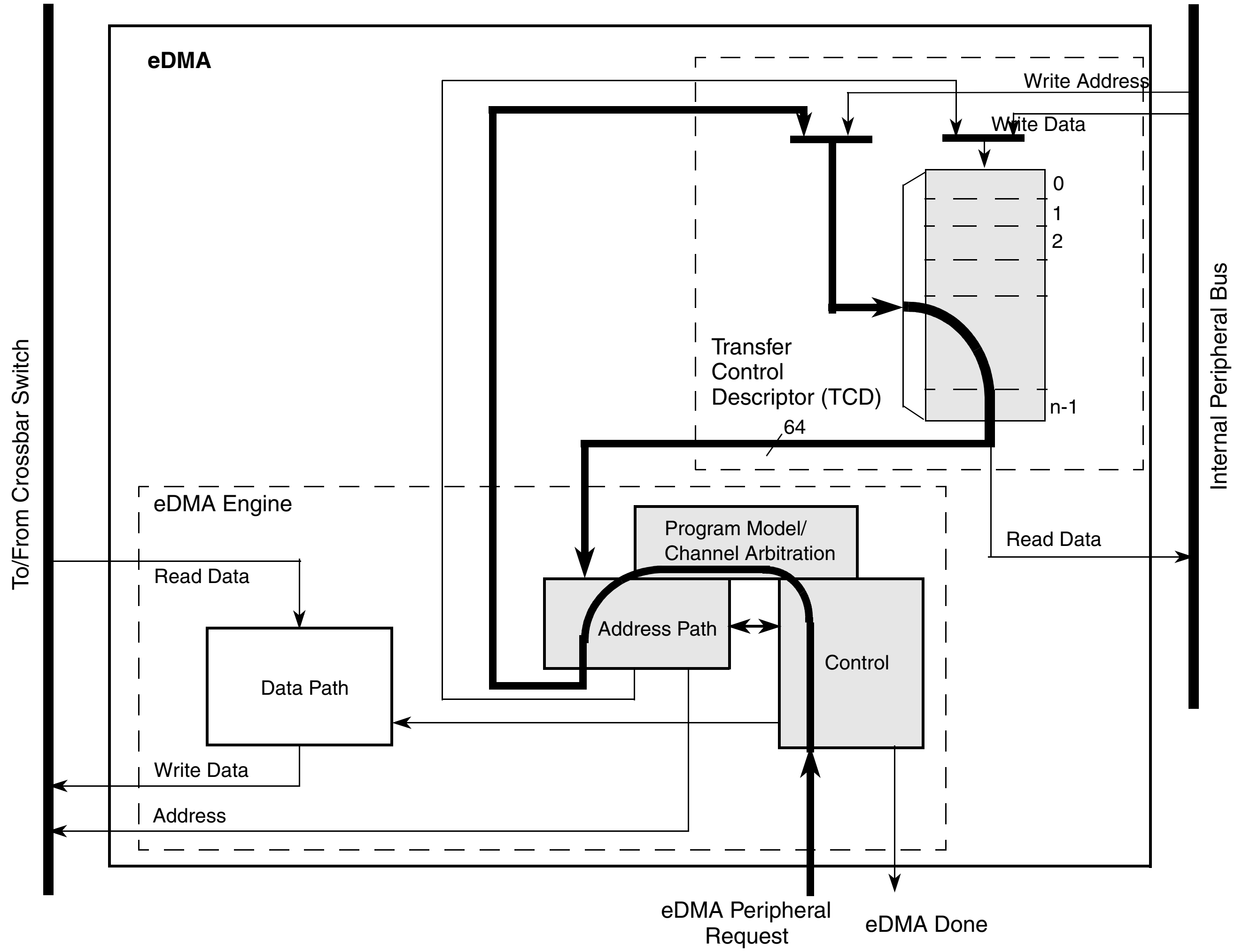


Figure 19-29. eDMA Operation, Part 1

In the second part of the basic data flow (Figure 19-30), the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path block until it is gated onto the internal bus during the destination write. This source read/destination write processing continues until the minor byte count has transferred.

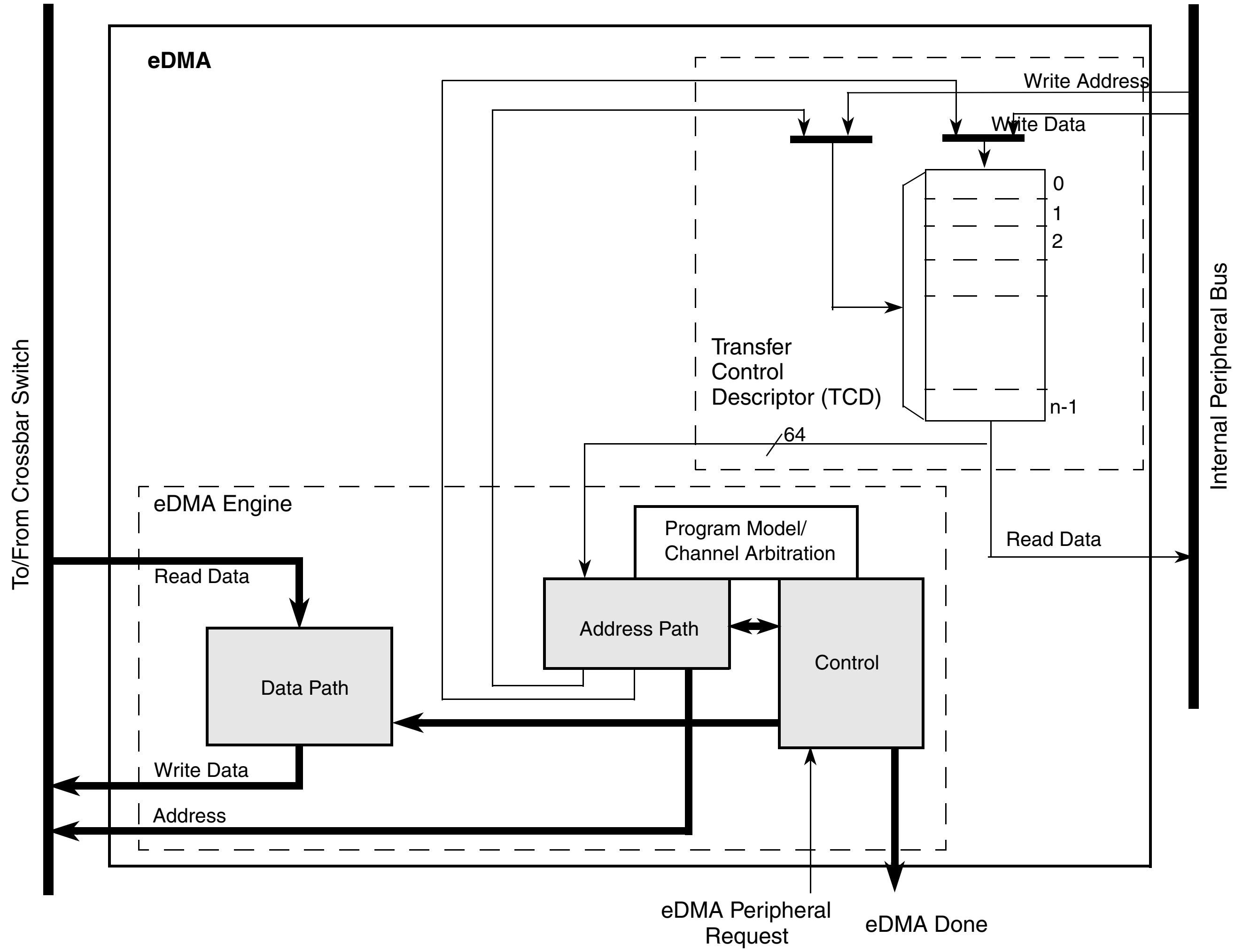


Figure 19-30. eDMA Operation, Part 2

After the minor byte count has moved, the final phase of the basic data flow performs. In this segment, the address path logic performs the required updates to certain fields in the appropriate TCD, e.g., SADDR, DADDR, CITER. If the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in Figure 19-31.

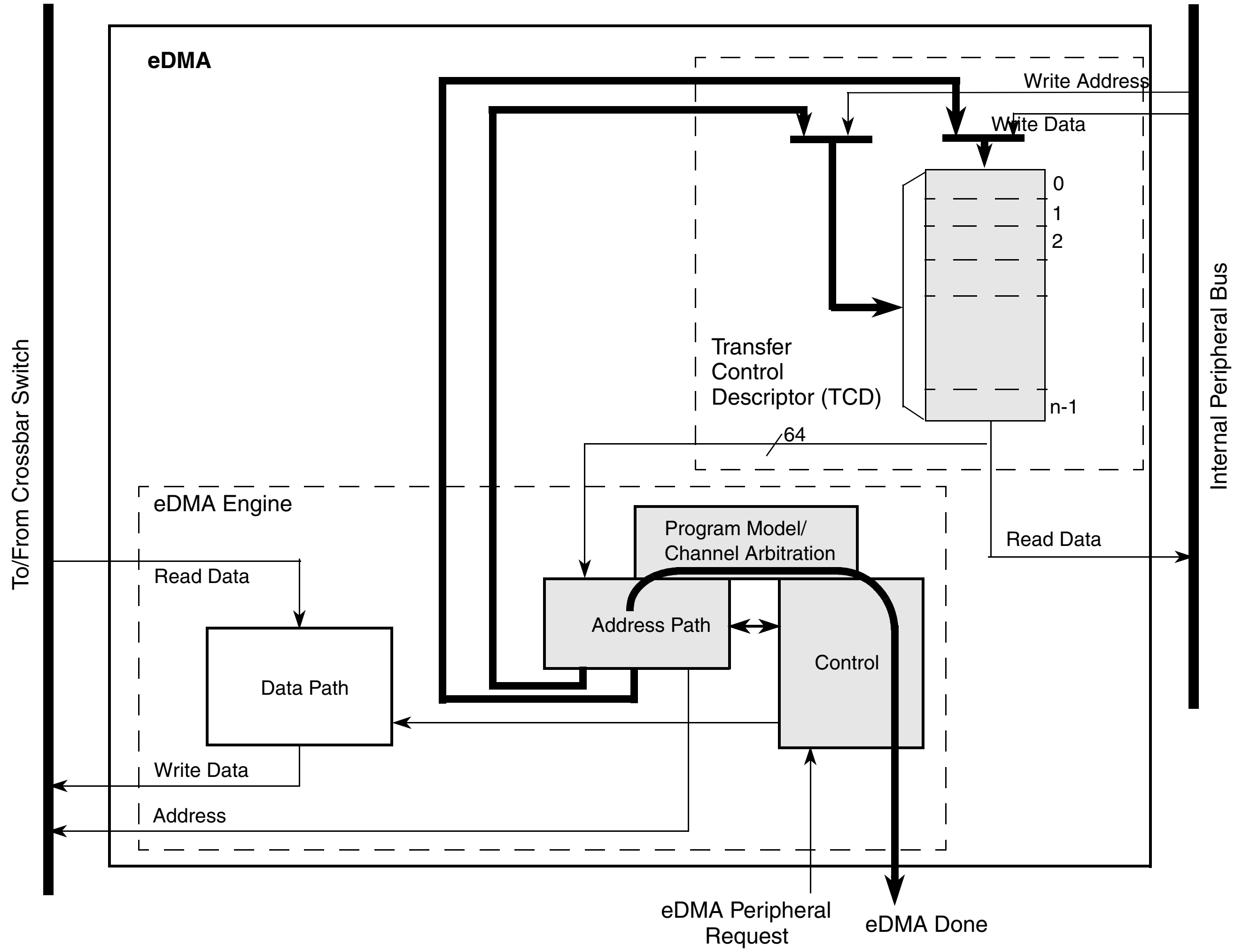


Figure 19-31. eDMA Operation, Part 3

19.6 Initialization/Application Information

19.6.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRI*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEI if so desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_ERQ.
6. Request channel service by software (setting the TCD*n*_CSR[START] bit) or hardware (slave device asserting its eDMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine read the entire TCD, including the TCD control and status fields (Table 19-31) for the selected channel into its internal address path module. As the TCD is read, the first transfer is initiated on the internal bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD*n*_SADDR) to the destination (as defined by the destination address, TCD*n*_DADDR) continue until the specified number of bytes (TCD*n*_NBYTES) are transferred. When transfer is complete, the eDMA engine's local TCD*n*_SADDR, TCD*n*_DADDR, and TCD*n*_CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing executes (interrupts, major loop channel linking, and scatter/gather operations) if enabled.

Table 19-31. TCD Control and Status Fields

TCD*n*_CSR Field Name	Description
START	Control bit to start channel explicitly when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

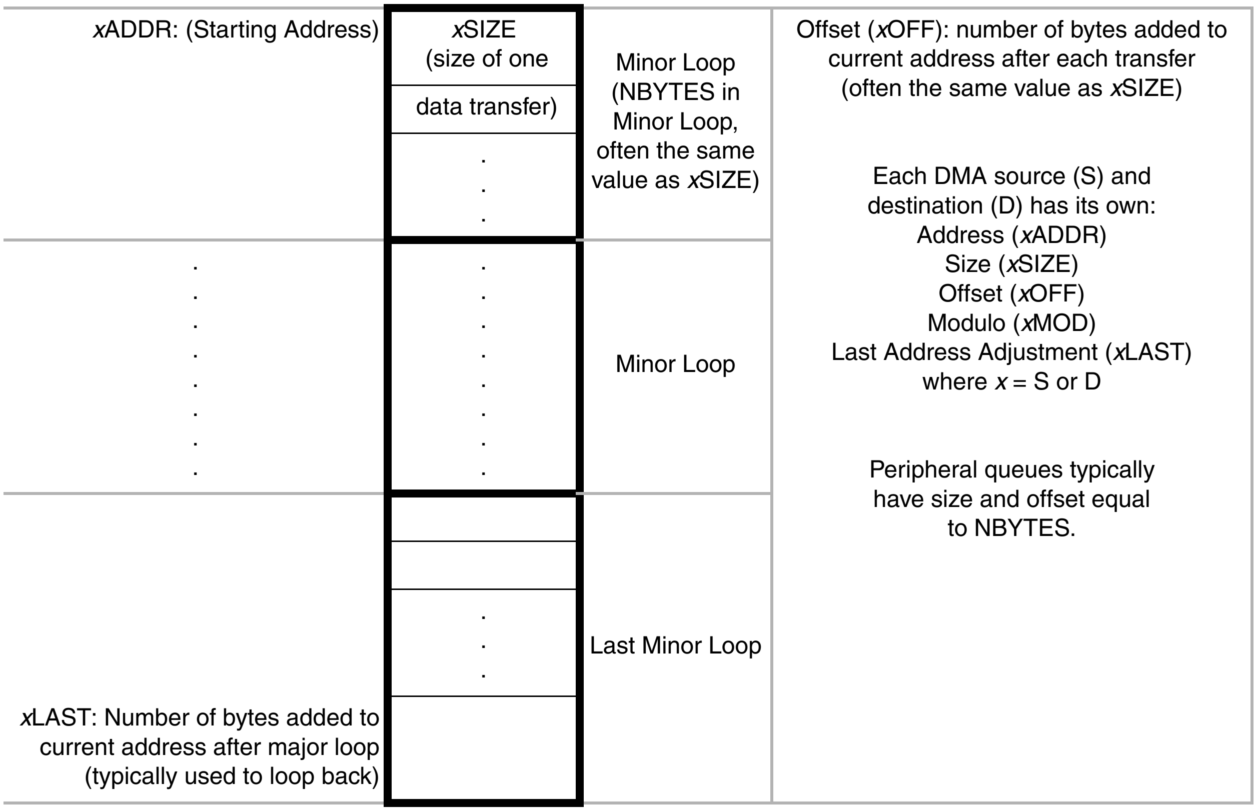
Table 19-32 shows how each DMA request initiates one minor-loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (BITER).

Table 19-32. Example of Multiple Loop Iterations

				Current Major Loop Iteration Count (CITER)
DMA Request	. . .	Minor Loop	Major Loop	3
DMA Request	. . .	Minor Loop		2
DMA Request	. . .	Minor Loop		1

Table 19-33 lists the memory array terms and how the TCD settings interrelate.

Table 19-33. Memory Array Terms


*x*ADDR: (Starting Address)
*x*SIZE (size of one data transfer)
Minor Loop (NBYTES in Minor Loop, often the same value as *x*SIZE)
Offset (*x*OFF): number of bytes added to current address after each transfer (often the same value as *x*SIZE)
Each DMA source (S) and destination (D) has its own:
Address (*x*ADDR)
Size (*x*SIZE)
Offset (*x*OFF)
Modulo (*x*MOD)
Last Address Adjustment (*x*LAST)
where *x* = S or D
Peripheral queues typically have size and offset equal to NBYTES.
Minor Loop
Last Minor Loop
*x*LAST: Number of bytes added to current address after major loop (typically used to loop back)

19.6.2 DMA Programming Errors

The eDMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of channel priority error (EDMA_ES[CPE]).

For all error types other than channel priority error, the channel number causing the error is recorded in the EDMA_ES. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

If priority levels are not unique, when any channel requests service, a channel priority error is reported. The highest channel priority with an active request is selected, but the lowest numbered channel with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts, and error reporting is associated with the selected channel.

19.6.3 DMA Arbitration Mode Considerations

19.6.3.1 Fixed Channel Arbitration

In this mode, the channel service request from the highest priority channel is selected to execute.

19.6.3.2 Round Robin Channel Arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels

19.6.4 DMA Transfer

19.6.4.1 Single Request

To perform a simple transfer of n bytes of data with one activation, set the major loop to one (TCD*n*_CITER = TCD*n*_BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD*n*_CSR[DONE] bit is set and an interrupt generates if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a longword-wide port located at 0x2000. The address offsets are programmed in increments to match the transfer size: one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

Example 19-1. Single Request DMA Transfer

```
TCDn_CITER = TCDn_BITER = 1
TCDn_NBYTES = 16
TCDn_SADDR = 0x1000
TCDn_SOFF = 1
TCDn_ATTR[SSIZE] = 0
TCDn_SLAST = -16
TCDn_DADDR = 0x2000
TCDn_DOFF = 4
TCDn_ATTR[DSIZE] = 2
TCDn_DLAST_SGA= -16
TCDn_CSR[INT_MAJ] = 1
TCDn_CSR[START] = 1 (Should be written last after all other fields have been initialized)
All other TCDn fields = 0
```

This generates the following event sequence:

1. User write to the TCD*n*_CSR[START] bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD*n*_CSR[DONE] = 0, TCD*n*_CSR[START] = 0, TCD*n*_CSR[ACTIVE] = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source-to-destination transfers are executed as follows:
   a) Read byte from location 0x1000, read byte from location 0x1001, read byte from 0x1002, read byte from 0x1003.

   b) Write longword to location 0x2000 → first iteration of the minor loop.
   c) Read byte from location 0x1004, read byte from location 0x1005, read byte from 0x1006, read byte from 0x1007.
   d) Write longword to location 0x2004 → second iteration of the minor loop.
   e) Read byte from location 0x1008, read byte from location 0x1009, read byte from 0x100A, read byte from 0x100B.
   f) Write longword to location 0x2008 → third iteration of the minor loop.
   g) Read byte from location 0x100C, read byte from location 0x100D, read byte from 0x100E, read byte from 0x100F.
   h) Write longword to location 0x200C → last iteration of the minor loop → major loop complete.
6. The eDMA engine writes: TCD*n*_SADDR = 0x1000, TCD*n*_DADDR = 0x2000, TCD*n*_CITER = 1 (TCD*n*_BITER).
7. The eDMA engine writes: TCD*n*_CSR[ACTIVE] = 0, TCD*n*_CSR[DONE] = 1, EDMA_INT[*n*] = 1.
8. The channel retires and the eDMA goes idle or services the next channel.

19.6.4.2 Multiple Requests

Besides transferring 32 bytes via two hardware requests, the next example is the same as previous. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in EDMA_ERQ, the slave device initiates channel service requests.

```
TCDn_CITER = TCDn_BITER = 2
TCDn_SLAST = -32
TCDn_DLAST_SGA = -32
```

This would generate the following sequence of events:

1. First hardware (eDMA peripheral) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD*n*_CSR[DONE] = 0, TCD*n*_CSR[START] = 0, TCD*n*_CSR[ACTIVE] = 1.
4. eDMA engine reads: channel TCD*n* data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) Read byte from location 0x1000, read byte from location 0x1001, read byte from 0x1002, read byte from 0x1003.
   b) Write longword to location 0x2000 → first iteration of the minor loop.
   c) Read byte from location 0x1004, read byte from location 0x1005, read byte from 0x1006, read byte from 0x1007.
   d) Write longword to location 0x2004 → second iteration of the minor loop.
   e) Read byte from location 0x1008, read byte from location 0x1009, read byte from 0x100A, read byte from 0x100B.

   f) Write longword to location 0x2008 → third iteration of the minor loop.
   g) Read byte from location 0x100C, read byte from location 0x100D, read byte from 0x100E, read byte from 0x100F.
   h) Write longword to location 0x200C → last iteration of the minor loop.
6. eDMA engine writes: TCD*n*_SADDR = 0x1010, TCD*n*_DADDR = 0x2010, TCD*n*_CITER = 1.
7. eDMA engine writes: TCD*n*_CSR[ACTIVE] = 0.
8. The channel retires → one iteration of the major loop. The eDMA goes idle or services the next channel.
9. Second hardware (eDMA peripheral) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD*n*_CSR[DONE] = 0, TCD*n*_CSR[START] = 0, TCD*n*_CSR[ACTIVE] = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
   a) Read byte from location 0x1010, read byte from location 0x1011, read byte from 0x1012, read byte from 0x1013.
   b) Write longword to location 0x2010 → first iteration of the minor loop.
   c) Read byte from location 0x1014, read byte from location 0x1015, read byte from 0x1016, read byte from 0x1017.
   d) Write longword to location 0x2014 → second iteration of the minor loop.
   e) Read byte from location 0x1018, read byte from location 0x1019, read byte from 0x101A, read byte from 0x101B.
   f) Write longword to location 0x2018 → third iteration of the minor loop.
   g) Read byte from location 0x101C, read byte from location 0x101D, read byte from 0x101E, read byte from 0x101F.
   h) Write longword to location 0x201C → last iteration of the minor loop → major loop complete.
14. eDMA engine writes: TCD*n*_SADDR = 0x1000, TCD*n*_DADDR = 0x2000, TCD*n*_CITER = 2 (TCD*n*_BITER).
15. eDMA engine writes: TCD*n*_CSR[ACTIVE] = 0, TCD*n*_CSR[DONE] = 1, EDMA_INT[n] = 1.
16. The channel retires → major loop complete. The eDMA goes idle or services the next channel.

19.6.4.3 Modulo Feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of 2. MOD is a 5-bit field for the source and destination in the TCD, and it specifies which lower address bits increment from their original value after the address+offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 19-34 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits

(0x1234567*x*) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the MOD field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 19-34. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

19.6.5 eDMA TCD*n* Status Monitoring

19.6.5.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first is to read the TCD*n*_CITER field and test for a change. (Another method may be extracted from the sequence shown below). The second method is to test the TCD*n*_CSR[START] bit and the TCD*n*_CSR[ACTIVE] bit. The minor-loop-complete condition is indicated by both bits reading zero after the TCD*n*_CSR[START] was set. Polling the TCD*n*_CSR[ACTIVE] bit may be inconclusive, because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

	TCD*n*_CSR bits			State
	START	ACTIVE	DONE	
1	1	0	0	Channel service request via software
2	0	1	0	Channel is executing
3a	0	0	0	Channel has completed the minor loop and is idle
3b	0	0	1	Channel has completed the major loop and is idle

The best method to test for minor-loop completion when using hardware (peripheral) initiated service requests is to read the TCD*n*_CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware-activated channel:

	TCD*n*_CSR bits			State
	START	ACTIVE	DONE	
1	0	0	0	Channel service request via hardware (peripheral request asserted)
2	0	1	0	Channel is executing
3a	0	0	0	Channel has completed the minor loop and is idle
3b	0	0	1	Channel has completed the major loop and is idle

For both activation types, the major-loop-complete status is explicitly indicated via the TCD*n*_CSR[DONE] bit.

The TCD*n*_CSR[START] bit is cleared automatically when the channel begins execution regardless of how the channel activates.

19.6.5.2 Active Channel TCD*n* Reads

The eDMA reads back the true TCD*n*_SADDR, TCD*n*_DADDR, and TCD*n*_NBYTES values if read while a channel executes. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine currently uses in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

19.6.5.3 Preemption Status

Preemption is available only when fixed arbitration is selected as the channel arbitration mode. A preemptive situation is one in which a preempt-enabled channel runs and a higher priority request becomes active. When the eDMA engine is not operating in fixed channel arbitration mode, the determination of the actively running relative priority outstanding requests become undefined. Channel priorities are treated as equal (constantly rotating) when round-robin arbitration mode is selected.

The TCD*n*_CSR[ACTIVE] bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one major loop iteration. If two TCD*n*_CSR[ACTIVE] bits are set simultaneously in the global TCD map, a higher priority channel is actively preempting a lower priority channel.

19.6.6 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD*n*_CSR[START] bit of another channel (or itself), therefore initiating a service request for that channel. When properly enabled, the EDMA engine automatically performs this operation at the major or minor loop completion.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD*n*_CITER[E_LINK] field determines whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, the initial fields of:

```
TCDn_CITER[E_LINK] = 1
TCDn_CITER[LINKCH] = 0xC
TCDn_CITER[CITER] value = 0x4
TCDn_CSR[MAJOR_E_LINK] = 1
TCDn_CSR[MAJOR_LINKCH] = 0x7
```

executes as:

1. Minor loop done → set TCD12_CSR[START] bit
2. Minor loop done → set TCD12_CSR[START] bit
3. Minor loop done → set TCD12_CSR[START] bit
4. Minor loop done, major loop done → set TCD7_CSR[START] bit

When minor loop linking is enabled (TCD*n*_CITER[E_LINK] = 1), the TCD*n*_CITER[CITER] field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled (TCD*n*_CITER[E_LINK] = 0), the TCD*n*_CITER[CITER] field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD*n*_CITER[LINKCH] field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

> The TCD*n*_CITER[E_LINK] bit and the TCD*n*_BITER[E_LINK] bit must equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, half-way done interrupt point.

Table 19-35 summarizes how a DMA channel can link to another DMA channel, i.e, use another channel's TCD, at the end of a loop.

Table 19-35. Channel Linking Parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of Minor Loop	CITER[E_LINK]	Enable channel-to-channel linking on minor loop completion (current iteration)
	CITER[LINKCH]	Link channel number when linking at end of minor loop (current iteration)
Link at end of Major Loop	CSR[MAJOR_E_LINK]	Enable channel-to-channel linking on major loop completion
	CSR[MAJOR_LINKCH]	Link channel number when linking at end of major loop

19.6.7 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

19.6.7.1 Dynamic Channel Linking and Dynamic Scatter/Gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD*n*_CSR[MAJOR_E_LINK] or TCD*n*_CSR[E_SG] bits during channel execution. These bits are read from the TCD local memory at the end of channel execution, therefore allowing software to enable either feature during channel execution.

Because software can change the configuration during execution, a coherency sequence must be followed. Consider the scenario the user attempts to execute a dynamic channel link by enabling the TCD*n*_CSR[MAJOR_E_LINK] bit as the eDMA engine retires the channel. The TCD*n*_CSR[MAJOR_E_LINK] would be set in the programmer's model, but it would be indeterminate whether the actual link was made before the channel retired.

The following coherency sequence is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD*n*_CSR[MAJOR_E_LINK] bit.
2. Read back the TCD*n*_CSR[MAJOR_E_LINK] bit.
3. Test the TCD*n*_CSR[MAJOR_E_LINK] request status.
   a) If the bit is set, the dynamic link attempt was successful.
   b) If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD*n*_CSR[MAJOR_E_LINK] and TCD*n*_CSR[E_SG] bits to zero on any writes to a TCD*n* after the TCD*n*_CSR[DONE] bit for that channel is set, indicating the major loop is complete.

NOTE

Software must clear the TCD*n*_CSR[DONE] bit before writing the TCD*n*_CSR[MAJOR_E_LINK] or TCD*n*_CSR[E_SG] bits. The TCD*n*_CSR[DONE] bit is cleared automatically by the eDMA engine after a channel begins execution.

Chapter 20
FlexBus

20.1 Introduction

This chapter describes external bus data transfer operations and error conditions. It describes transfers initiated by the ColdFire processor (or any other bus master) and includes detailed timing diagrams showing the interaction of signals in supported bus operations.

NOTE

- In this chapter, unless otherwise noted, clock refers to the FB_CLK used for the external bus ($f_{sys/4}$).
- When the MCF54450 and MCF54451 devices operate in non-multiplexed mode, only a 24-bit external address is available, FB_A[23:0]

20.1.1 Overview

A multi-function external bus interface called the FlexBus interface controller is provided on the device with basic functionality of interfacing to slave-only devices. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry:

- External boot ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

For asynchronous devices, a simple chip-select based interface can be used.

The FlexBus interface has up to six general purpose chip-selects, $\overline{\text{FB\_CS}}$[5:0]. The actual number of chip selects available depends upon the device and its pin configuration. Chip-select $\overline{\text{FB\_CS0}}$ can be dedicated to boot memory access and programmed to be byte (8 bits), word (16 bits), or longword (32 bits) wide. Control signal timing is compatible with common ROM and flash memories.

20.1.2 Features

Key FlexBus features include:

- Six independent, user-programmable chip-select signals ($\overline{\text{FB\_CS}}$[5:0]) that can interface with external SRAM, PROM, EPROM, EEPROM, flash, and other peripherals
- 8-, 16-, and 32-bit port sizes with configuration for multiplexed or non-multiplexed address and data buses

- Byte-, word-, longword-, and 16-byte line-sized transfers
- Programmable burst- and burst-inhibited transfers selectable for each chip select and transfer direction
- Programmable address-setup time with respect to the assertion of chip select
- Programmable address-hold time with respect to the negation of chip select and transfer direction

20.1.3 Modes of Operation

The external interface is a configurable multiplexed bus set to one of the following modes:

- Multiplexed 32-bit address and 32-bit data
- Multiplexed 32-bit address and 16-bit data (non-multiplexed 16-bit address and 16-bit data)
- Multiplexed 32-bit address and 8-bit data (non-multiplexed 24-bit address and 8-bit data)
- Non-multiplexed 32-bit address with 32-bit data

NOTE

While using non-multiplexed 32-bit mode, the PCI address/data bus is used as the FlexBus address bus. See Chapter 11, “Chip Configuration Module (CCM),” for more information.

20.2 External Signals

This section describes the external signals involved in data-transfer operations.

Table 20-1. FlexBus Signal Summary

Signal Name	I/O[1]	Description
FB_A[31:0]	O	In a non-multiplexed configuration: Address bus. In a multiplexed configuration: Not used.
FB_D[31:0]	I/O	In a non-multiplexed configuration: Data bus. In a multiplexed configuration: Address/data bus, FB_AD[31:0].
$\overline{\text{FB\_CS}}$[5:0]	O	General purpose chip-selects. The actual number of chip selects available depends upon the device and its pin configuration. See Table 1-1Table 2-2 for more details.
$\overline{\text{FB\_BE/BWE}}$[3:0]	O	Byte enable/byte write enable
$\overline{\text{FB\_OE}}$	O	Output enable
FB_R/$\overline{\text{W}}$	O	Read/write. 1 = Read, 0 = Write
FB_ALE	O	Address latch enable
FB_TSIZ[1:0]	O	Transfer size
$\overline{\text{FB\_TBST}}$	O	Burst transfer indicator
$\overline{\text{FB\_TA}}$	I	Transfer acknowledge

[1] Because this device shares the FlexBus signals with the PCI controller, these signal directions are only valid when the FlexBus controls them. The directions may change during PCI cycles.

20.2.1 Address and Data Buses (FB_A*n*, FB_D*n*, FB_AD*n*)

In non-multiplexed mode, the FB_A*n* and FB_D*n* buses carry the address and data, respectively. The number of byte lanes carrying the data is determined by the port size associated with the matching chip select.

In multiplexed mode, the FB_AD*n* bus carries the address and data. The full 32 -bit address is driven on the first clock of a bus cycle (address phase). Following the first clock, the data is driven on the bus (data phase). During the data phase, the address continues driving on the pins not used for data. For example, in 16-bit mode the address continues driving on FB_AD[15 :0] and in 8-bit mode the address continues driving on FB_AD[23:0].

In non-multiplexed mode, the FB_AD[31:0] signals carry the data and the PCI address/data bus carries the address. As a result, the PCI is disabled when the FlexBus operates in non-multiplexed mode.

Multiplexed/non-multiplexed operation is determined at reset by FB_AD[7:5]. See Chapter 11, "Chip Configuration Module (CCM)," for more information.

Because this device shares the FlexBus signals with the PCI controller, these signals tristate between bus cycles.

20.2.2 Chip Selects (FB_CS[5:0])

The chip-select signal indicates which device is selected. A particular chip-select asserts when the transfer address is within the device's address space, as defined in the base- and mask-address registers. The actual number of chip selects available depends upon the pin configuration.

20.2.3 Byte Enables/Byte Write Enables (FB_BE/BWE[3:0])

When driven low, the byte enable (FB_BE/BWE[3:0]) outputs indicate data is to be latched or driven onto a byte of the data bus. FB_BE/BWE*n* signals are asserted only to the memory bytes used during read or write accesses. A configuration option is provided to assert these signals on reads and writes (byte enable) or writes only (byte-write enable).

The FB_BE/BWE*n* signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM or cache. For external SRAM or flash devices, the FB_BE/BWE*n* outputs must be connected to individual byte strobe signals.

20.2.4 Output Enable (FB_OE)

The output enable signal (FB_OE) is sent to the interfacing memory and/or peripheral to enable a read transfer. FB_OE is only asserted during read accesses when a chip select matches the current address decode.

Because this device shares the FlexBus signals with the PCI controller, this signal tristates between bus cycles.

20.2.5 Read/Write (FB_R/$\overline{\mathrm{W}}$)

The processor drives the FB_R/$\overline{\mathrm{W}}$ signal to indicate the current bus operation direction. It is driven high during read bus cycles and low during write bus cycles.

Because this device shares the FlexBus signals with the PCI controller, this signal tristates between bus cycles.

20.2.6 Address Latch Enable (FB_ALE)

The assertion of FB_ALE indicates that the device has begun a bus transaction and the address and attributes are valid. FB_ALE is asserted for one bus clock cycle. FB_ALE may be used externally to capture the bus transfer address (Figure 20-7).

Because this device shares the FlexBus signals with the PCI controller, this signal tristates between bus cycles.

20.2.7 Transfer Size (FB_TSIZ[1:0])

For memory accesses, these signals, along with $\overline{\mathrm{FB\_TBST}}$, indicate the data transfer size of the current bus operation. The interface supports byte, word, and longword operand transfers and allows accesses to 8-, 16-, and 32-bit data ports.

For misaligned transfers, FB_TSIZ[1:0] indicates the size of each transfer. For example, if a longword access through a 32-bit port device occurs at a misaligned offset of 0x1, a byte is transferred first (FB_TSIZ[1:0] = 01), a word is transferred next at offset 0x2 (FB_TSIZ[1:0] = 10), and the final byte is transferred at offset 0x4 (FB_TSIZ[1:0] = 01).

For aligned transfers larger than the port size, FB_TSIZ[1:0] behaves as follows:

- If bursting is used, FB_TSIZ[1:0] is driven to the transfer size.
- If bursting is inhibited, FB_TSIZ[1:0] first shows the entire transfer size and then shows the port size.

Table 20-2. Data Transfer Size

FB_TSIZ[1:0]	Transfer Size
00	4 bytes (longword)
01	1 byte
10	2 bytes (word)
11	16 bytes (line)

For burst-inhibited transfers, FB_TSIZ[1:0] changes with each $\overline{\mathrm{FB\_TS}}$ assertion to reflect the next transfer size. For transfers to port sizes smaller than the transfer size, FB_TSIZ[1:0] indicates the size of the entire transfer on the first access and the size of the current port transfer on subsequent transfers. For example, for a longword write to an 8-bit port, FB_TSIZ[1:0] equals 00 for the first transaction and 01 for the next three transactions. If bursting is used for longword write to an 8-bit port, FB_TSIZ[1:0] is driven to 00 for the entire transfer.

20.2.8 Transfer Burst (FB_TBST)

Transfer burst indicates that a burst transfer is in progress as driven by the device. A burst transfer can be two to 16 beats depending on FB_TSIZ[1:0] and the port size.

NOTE

When burst (FB_TBST = 0), transfer size is 16 bytes (FB_TSIZ[1:0] = 11) and the address is misaligned within the 16-byte boundary, the external device must be able to wrap around the address.

20.2.9 Transfer Acknowledge (FB_TA)

This signal indicates the external data transfer is complete. When the processor recognizes FB_TA during a read cycle, it latches the data and then terminates the bus cycle. When the processor recognizes FB_TA during a write cycle, the bus cycle is terminated.

If auto-acknowledge is disabled (CSCR*n*[AA] = 0), the external device drives FB_TA to terminate the bus transfer; if auto-acknowledge is enabled (CSCR*n*[AA] = 1), FB_TA is generated internally after a specified number of wait states, or the external device may assert external FB_TA before the wait-state countdown, terminating the cycle early. The device negates FB_CS*n* one cycle after the last FB_TA asserts. During read cycles, the peripheral must continue to drive data until FB_TA is recognized. For write cycles, the processor continues driving data one clock after FB_CS*n* is negated.

The number of wait states is determined by CSCR*n* or the external FB_TA input. If the external FB_TA is used, the peripheral has total control on the number of wait states.

NOTE

External devices should only assert FB_TA while the FB_CS*n* signal to the external device is asserted.

Because this device shares the FlexBus signals with the PCI controller, this signal tristates between bus cycles.

20.3 Memory Map/Register Definition

The following tables describe the registers and bit meanings for configuring chip-select operation. Table 20-3 shows the chip-select register memory map.

The actual number of chip select registers available depends upon the device and its pin configuration. If the device does not support certain chip select signals or the pin is not configured for a chip-select function, then that corresponding set of chip-select registers has no effect on an external pin.

NOTE

You must set CSMR0[V] before the chip select registers take effect.

Table 20-3. FlexBus Chip Select Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC00_8000 + (*n* × 0xC)	Chip-Select Address Register (CSAR*n*) *n* = 0 – 5	32	R/W	0x0000_0000	20.3.1/20-6
0xFC00_8004 + (*n* × 0xC)	Chip-Select Mask Register (CSMR*n*) *n* = 0 – 5	32	R/W	0x0000_0000	20.3.2/20-7
0xFC00_8008 + (*n* × 0xC)	Chip-Select Control Register (CSCR*n*) *n* = 0 – 5	32	R/W	See Section	20.3.3/20-7

20.3.1 Chip-Select Address Registers (CSAR0 – CSAR5)

The CSAR*n* registers specify the chip-select base addresses.

NOTE

Because the FlexBus module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x0000_0000 – 0x3FFF_FFFF and 0xC000_0000 – 0xDFFF_FFFF. Set the CSAR*n* registers appropriately.

Address: 0xFC00_8000 (CSAR0)
0xFC00_800C (CSAR1)
0xFC00_8018 (CSAR2)
0xFC00_8024 (CSAR3)
0xFC00_8030 (CSAR4)
0xFC00_803C (CSAR5)

Access: User 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
R	BA																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-1. Chip-Select Address Registers (CSAR*n*)

Table 20-4. CSAR*n* Field Descriptions

Field	Description
31–16 BA	Base address. Defines the base address for memory dedicated to chip-select $\overline{\text{FB\_CS}n}$. BA is compared to bits 31–16 on the internal address bus to determine if chip-select memory is being accessed.
15–0	Reserved, must be cleared.

20.3.2 Chip-Select Mask Registers (CSMR0 – CSMR5)

CSMR*n* registers specify the address mask and allowable access types for the respective chip-selects.

Address: 0xFC00_8004 (CSMR0)
0xFC00_8010 (CSMR1)
0xFC00_801C (CSMR2)
0xFC00_8028 (CSMR3)
0xFC00_8034 (CSMR4)
0xFC00_8040 (CSMR5)

Access: User 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
R	BAM																0	0	0	0	0	0	0	WP	0	0	0	0	0	0	0	V
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-2. Chip-Select Mask Registers (CSMR*n*)

Table 20-5. CSMR*n* Field Descriptions

Field	Description
31–16 BAM	Base address mask. Defines the chip-select block size by masking address bits. Setting a BAM bit causes the corresponding CSAR bit to be a don't care in the decode. 0 Corresponding address bit is used in chip-select decode. 1 Corresponding address bit is a don't care in chip-select decode. The block size for FB_CS*n* is 2^n; n = (number of bits set in respective CSMR[BAM]) + 16. For example, if CSAR0 equals 0x0000 and CSMR0[BAM] equals 0x0008, FB_CS0 addresses two discontinuous 64 KB memory blocks: one from 0x0_0000 – 0x0_FFFF and one from 0x8_0000 – 0x8_FFFF. Likewise, for FB_CS0 to access 32 MB of address space starting at location 0x00_0000, FB_CS1 must begin at the next byte after FB_CS0 for a 16 MB address space. Therefore, CSAR0 equals 0x0000, CSMR0[BAM] equals 0x01FF, CSAR1 equals 0x0200, and CSMR1[BAM] equals 0x00FF.
15–9	Reserved, must be cleared.
8 WP	Write protect. Controls write accesses to the address range in the corresponding CSAR. Attempting to write to the range of addresses for which CSAR*n*[WP] is set results in a bus error termination of the internal cycle and no external cycle. 0 Read and write accesses are allowed 1 Only read accesses are allowed
7–1	Reserved, must be cleared.
0 V	Valid bit. Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. Programmed chip-selects do not assert until V bit is set (except for FB_CS0, which acts as the global chip-select). Reset clears each CSMR*n*[V]. **Note:** At reset, no chip-select other than FB_CS0 can be used until the CSMR0[V] is set. Afterward, FB_CS[5:0] functions as programmed. 0 Chip-select invalid 1 Chip-select valid

20.3.3 Chip-Select Control Registers (CSCR0 – CSCR5)

Each CSCR*n* controls the auto-acknowledge, address setup and hold times, port size, burst capability, and number of wait states. To support the global chip-select, FB_CS0, the CSCR0 reset values differ from the other CSCRs. $\overline{\text{FB\_CS0}}$ allows address decoding for an external device to serve as the boot memory before system initialization and configuration are completed.

Address: 0xFC00_8008 (CSCR0)
0xFC00_8014 (CSCR1)
0xFC00_8020 (CSCR2)
0xFC00_802C (CSCR3)
0xFC00_8038 (CSCR4)
0xFC00_8044 (CSCR5)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SWS						0	0	SWSEN	0	ASET		RDAH		WRAH	
W																
Reset: CSCR0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
Reset: CSCR1–5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WS						0	AA	PS		BEM	BSTR	BSTW	0	0	0
W																
Reset: CSCR0	1	1	1	1	1	1	0	1	See Note	See Note	1	0	0	0	0	0
Reset: CSCR1–5	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0

Note: The PS reset value depends upon the chosen chip configuration (RCON[7:5] for parallel configuration or SBF_RCON[127:126] for serial boot configuration).

Figure 20-3. Chip-Select Control Registers (CSCR*n*)

Table 20-6. CSCR*n* Field Descriptions

Field	Description
31–26 SWS	Secondary wait states. The number of wait states inserted before an internal transfer acknowledge is generated for a burst transfer except for the first termination, which is controlled by the wait state count. The secondary wait state is used only if the SWSEN bit is set. Otherwise, the WS value is used for all burst transfers.
25–24	Reserved, must be cleared
23 SWSEN	Secondary wait state enable. 0 The WS value inserts wait states before an internal transfer acknowledge is generated for all transfers. 1 The SWS value inserts wait states before an internal transfer acknowledge is generated for burst transfer secondary terminations.
22	Reserved, must be cleared
21–20 ASET	Address setup. This field controls the assertion of the chip-select with respect to assertion of a valid address and attributes. The address and attributes are considered valid at the same time FB_ALE asserts. 00 Assert $\overline{\text{FB\_CS}n}$ on first rising clock edge after address is asserted. (Default $\overline{\text{FB\_CS}n}$) 01 Assert $\overline{\text{FB\_CS}n}$ on second rising clock edge after address is asserted. 10 Assert $\overline{\text{FB\_CS}n}$ on third rising clock edge after address is asserted. 11 Assert $\overline{\text{FB\_CS}n}$ on fourth rising clock edge after address is asserted. (Default $\overline{\text{FB\_CS0}}$)

Table 20-6. CSCR*n* Field Descriptions (Continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>19–18
RDAH</td><td>Read address hold or deselect. This field controls the address and attribute hold time after the termination during a read cycle that hits in the chip-select address space.
<b>Note:</b> The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.
The number of cycles the address and attributes are held after FB_CS<i>n</i> negation depends on the value of CSCR<i>n</i>[AA] as shown below.
<table>
<tr><th>RDAH</th><th>AA = 0</th><th>AA = 1</th></tr>
<tr><td>00
(FB_CS<i>n</i> Default)</td><td>1 cycle</td><td>0 cycles</td></tr>
<tr><td>01</td><td>2 cycles</td><td>1 cycles</td></tr>
<tr><td>10</td><td>3 cycles</td><td>2 cycles</td></tr>
<tr><td>11
(FB_CS0 Default)</td><td>4 cycles</td><td>3 cycles</td></tr>
</table>
</td></tr>
<tr><td>17–16
WRAH</td><td>Write address hold or deselect. This field controls the address, data, and attribute hold time after the termination of a write cycle that hits in the chip-select address space.
<b>Note:</b> The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.
00 Hold address and attributes one cycle after FB_CS<i>n</i> negates on writes. (Default FB_CS<i>n</i>)
01 Hold address and attributes two cycles after FB_CS<i>n</i> negates on writes.
10 Hold address and attributes three cycles after FB_CS<i>n</i> negates on writes.
11 Hold address and attributes four cycles after FB_CS<i>n</i> negates on writes. (Default FB_CS0)</td></tr>
<tr><td>15–10
WS</td><td>Wait states. The number of wait states inserted after FB_CS<i>n</i> asserts and before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0x3F inserts 63 wait states). If AA is reserved, FB_TA must be asserted by the external system regardless of the number of generated wait states. In that case, the external transfer acknowledge ends the cycle. An external FB_TA supersedes the generation of an internal FB_TA.</td></tr>
<tr><td>9</td><td>Reserved, must be cleared.</td></tr>
<tr><td>8
AA</td><td>Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address.
0 No internal FB_TA is asserted. Cycle is terminated externally
1 Internal transfer acknowledge is asserted as specified by WS

<b>Note:</b> If AA is set for a corresponding FB_CS<i>n</i> and the external system asserts an external FB_TA before the wait-state countdown asserts the internal FB_TA, the cycle is terminated. Burst cycles increment the address bus between each internal termination.</td></tr>
<tr><td>7–6
PS</td><td>Port size. Specifies the data port width associated with each chip-select. It determines where data is driven during write cycles and where data is sampled during read cycles.
00 32-bit port size. Valid data sampled and driven on FB_D[31:0]
01 8-bit port size. Valid data sampled and driven on FB_D[31:24]
1<i>x</i> 16-bit port size. Valid data sampled and driven on FB_D[31:16]</td></tr>
</table>

Table 20-6. CSCR*n* Field Descriptions (Continued)

Field	Description
5 BEM	Byte-enable mode. Specifies the byte enable operation. Certain memories have byte enables that must be asserted during reads and writes. BEM can be set in the relevant CSCR to provide the appropriate mode of byte enable support for these SRAMs. 0 FB_BE/BWE is not asserted for reads. FB_BE/BWE is asserted for data write only. 1 FB_BE/BWE is asserted for read and write accesses.
4 BSTR	Burst-read enable. Specifies whether burst reads are used for memory associated with each FB_CS*n*. 0 Data exceeding the specified port size is broken into individual, port-sized, non-burst reads. For example, a longword read from an 8-bit port is broken into four 8-bit reads. 1 Enables data burst reads larger than the specified port size, including longword reads from 8- and 16-bit ports, word reads from 8-bit ports, and line reads from 8, 16-, and 32-bit ports.
3 BSTW	Burst-write enable. Specifies whether burst writes are used for memory associated with each FB_CS*n*. 0 Break data larger than the specified port size into individual, port-sized, non-burst writes. For example, a longword write to an 8-bit port takes four byte writes. 1 Enables burst write of data larger than the specified port size, including longword writes to 8 and 16-bit ports, word writes to 8-bit ports, and line writes to 8-, 16-, and 32-bit ports.
2–0	Reserved, must be cleared.

20.4 Functional Description

20.4.1 Chip-Select Operation

Each chip-select has a dedicated set of registers for configuration and control:

- Chip-select address registers (CSAR*n*) control the base address space of the chip-select. See Section 20.3.1, “Chip-Select Address Registers (CSAR0 – CSAR5).”
- Chip-select mask registers (CSMR*n*) provide 16-bit address masking and access control. See Section 20.3.2, “Chip-Select Mask Registers (CSMR0 – CSMR5).”
- Chip-select control registers (CSCR*n*) provide port size and burst capability indication, wait-state generation, address setup and hold times, and automatic acknowledge generation features. See Section 20.3.3, “Chip-Select Control Registers (CSCR0 – CSCR5).”

FB_CS0 is a global chip-select after reset and provides external boot memory capability.

20.4.1.1 General Chip-Select Operation

When a bus cycle is routed to the FlexBus, the device first compares its address with the base address and mask configurations programmed for chip-selects 0 to 5 (configured in CSCR0 – CSCR5). The results depend on if the address matches or not as shown in Table 20-7.

Table 20-7. Results of Address Comparison

Address Matches CSAR*n*?	Result
Yes, one CSAR	The appropriate chip-select is asserted, generating an external bus cycle as defined in the chip-select control register. If CSMR[WP] is set and a write access is performed, the internal bus cycle terminates with a bus error, no chip select is asserted, and no external bus cycle is performed.
No	The chip-select signals are not driven. However, the FlexBus runs an external bus cycle with external termination.
Yes, multiple CSARs	The chip-select signals are driven. However, they are driven using an external burst-inhibited bus cycle with external termination on a 32-bit port.

20.4.1.2 8-, 16-, and 32-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. The processor always drives a 32 -bit address on the FB_AD bus regardless of the external device's address size. The external device must connect its address lines to the appropriate FB_AD bits from FB_AD0 upward. Its data bus must be connected to FB_AD[7:0] from FB_AD31 downward. No bit ordering is required when connecting address and data lines to the FB_AD bus. For example, a full 16-bit address/16-bit data device connects its addr[15:0] to FB_AD[16:1] and data[15:0] to FB_AD[31:16]. See Figure 20-4 for a graphical connection.

20.4.1.3 Global Chip-Select Operation

$\overline{\text{FB\_CS0}}$, the global (boot) chip-select, supports external boot memory accesses before system initialization. Its operation differs from other external chip-select outputs after system reset.

After system reset, $\overline{\text{FB\_CS0}}$ is asserted for every external access. No other chip-select can be used until the valid bit, CSMR0[V], is set; at this point $\overline{\text{FB\_CS0}}$ functions as configured. After this, $\overline{\text{FB\_CS}}$[5:1] can be used as well. At reset during parallel boot, the logic levels on the FB_AD[4:3] signals determine global chip-select port size. During serial boot, the value of SBF_RCON[127:126] determine the port size.

See Chapter 11, "Chip Configuration Module (CCM)," for more information.

20.4.2 Data Transfer Operation

Data transfers between the chip and other devices involve these signals:

- Address/data bus (FB_AD[31:0])

- Control signals (FB_ALE, FB_TA, FB_CS*n*, FB_OE, FB_BE/BWE[3:0])
- Attribute signals (FB_R/W, FB_TBST, FB_TSIZ[1:0])

The address, write data, FB_ALE, FB_CS*n,* and all attribute signals change on the rising edge of the FlexBus clock (FB_CLK). Read data is latched into the device on the rising edge of the clock.

The FlexBus supports byte-, word-, longword-, and 16-byte (line) operand transfers and allows accesses to 8-, 16-, and 32-bit data ports.Transfer parameters (address setup and hold, port size, the number of wait states for the external device being accessed, automatic internal transfer termination enable or disable, and burst enable or disable) are programmed in the chip-select control registers (CSCRs). See Section 20.3.3, “Chip-Select Control Registers (CSCR0 – CSCR5).”

20.4.3 Data Byte Alignment and Physical Connections

The device aligns data transfers in FlexBus byte lanes with the number of lanes depending on the data port width. Figure 20-4 shows the byte lanes that external memory connects to and the sequential transfers of a longword transfer for the supported port sizes . For example, an 8-bit memory connects to the single lane FB_AD[31:24] (FB_BE/BWE0). A longword transfer through this 8-bit port takes four transfers, starting with the MSB to the LSB. A longword transfer through a 32-bit port requires one transfer on each four-byte lane of the FlexBus.

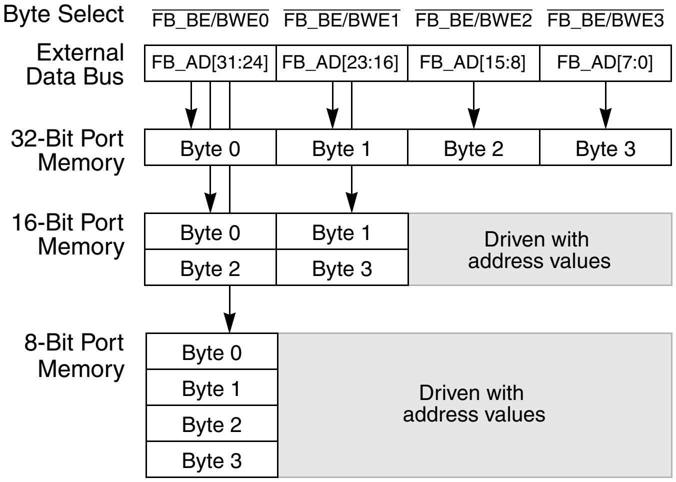


Figure 20-4. Connections for External Memory Port Sizes

20.4.4 Address/Data Bus Multiplexing

The interface supports a single 32-bit wide multiplexed address and data bus (FB_AD[31 :0]). The full 32-bit address is always driven on the first clock of a bus cycle. During the data phase, the FB_AD[31:0] lines used for data are determined by the programmed port size for the corresponding chip select. The device continues to drive the address on any FB_AD[31:0] lines not used for data.

The table below lists the supported combinations of address and data bus widths.

Table 20-8. FlexBus Multiplexed Operating Modes

Port Size and Phase		FB_AD			
		[31:24]	[23:16]	[15:8]	[7:0]
32-bit	Address phase	Address			
	Data phase	Data			
16-bit	Address phase	Address			
	Data phase	Data		Address	
8-bit	Address phase	Address			
	Data phase	Data	Address		

20.4.5 Bus Cycle Execution

As shown in Figure 20-7 and Figure 20-9, basic bus operations occur in four clocks:

1. S0: At the first clock edge, the address, attributes, and FB_ALE are driven.
2. S1: $\overline{\mathrm{FB\_CS}n}$ is asserted at the second rising clock edge to indicate the device selected; by that time, the address and attributes are valid and stable. FB_ALE is negated at this edge.

   For a write transfer, data is driven on the bus at this clock edge and continues to be driven until one clock cycle after $\overline{\mathrm{FB\_CS}n}$ negates. For a read transfer, data is also driven into the device during this cycle.

   External slave asserts $\overline{\mathrm{FB\_TA}}$ at this clock edge.
3. S2: Read data and $\overline{\mathrm{FB\_TA}}$ are sampled on the third clock edge. $\overline{\mathrm{FB\_TA}}$ can be negated after this edge and read data can then be tri-stated.
4. S3: $\overline{\mathrm{FB\_CS}n}$ is negated at the fourth rising clock edge. This last clock of the bus cycle uses what would be an idle clock between cycles to provide hold time for address, attributes, and write data.

20.4.5.1 Data Transfer Cycle States

An on-chip state machine controls the data-transfer operation in the device. Figure 20-5 shows the state-transition diagram for basic read and write cycles.

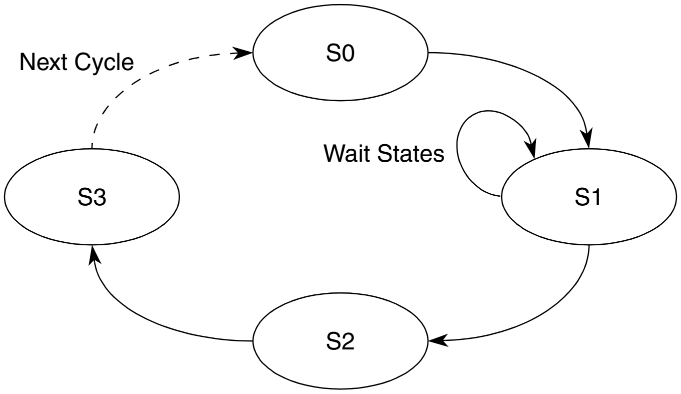


Figure 20-5. Data-Transfer-State-Transition Diagram

Table 20-10 describes the states as they appear in subsequent timing diagrams.

Table 20-10. Bus Cycle States

State	Cycle	Description
S0	All	The read or write cycle is initiated. On the rising clock edge, the device places a valid address on FB_AD[^{31} :0], asserts FB_ALE, and drives FB_R/$\overline{\text{W}}$ high for a read and low for a write.
S1	All	FB_ALE is negated on the rising edge of FB_CLK, and $\overline{\text{FB\_CS}n}$ is asserted. Data is driven on FB_AD[31:*X*] for writes, and FB_AD[31:*X*] is tristated for reads. Address continues to be driven on the FB_AD pins that are unused for data. If $\overline{\text{FB\_TA}}$ is recognized asserted, then the cycle moves on to S2. If $\overline{\text{FB\_TA}}$ is not asserted internally or externally, then the S1 state continues to repeat.
	Read	Data is driven by the external device before the next rising edge of FB_CLK (the rising edge that begins S2) with $\overline{\text{FB\_TA}}$ asserted.
S2	All	For internal termination, $\overline{\text{FB\_CS}n}$ is negated and the internal system bus transfer is completed. For external termination, the external device should negate $\overline{\text{FB\_TA}}$, and the $\overline{\text{FB\_CS}n}$ chip select negates after the rising edge of FB_CLK at the end of S2.
	Read	The processor latches data on the rising clock edge entering S2. The external device can stop driving data after this edge. However, data can be driven until the end of S3 or any additional address hold cycles.
S3	All	Address, data, and FB_R/$\overline{\text{W}}$ go invalid off the rising edge of FB_CLK at the beginning of S3, terminating the read or write cycle.

20.4.6 FlexBus Timing Examples

NOTE

Because this device shares the FlexBus signals with the PCI controller, all signals, except the chip selects, tristate between bus cycles.

20.4.6.1 Basic Read Bus Cycle

During a read cycle, the ColdFire device receives data from memory or a peripheral device. Figure 20-6 is a read cycle flowchart.

NOTE

Throughout this chapter FB_AD[31:*X*] indicates a 32-, 16-, or 8-bit wide data bus. FB_AD[*Y*:0] is an address bus that can be 32-, 24-, or 16 -bits in width.

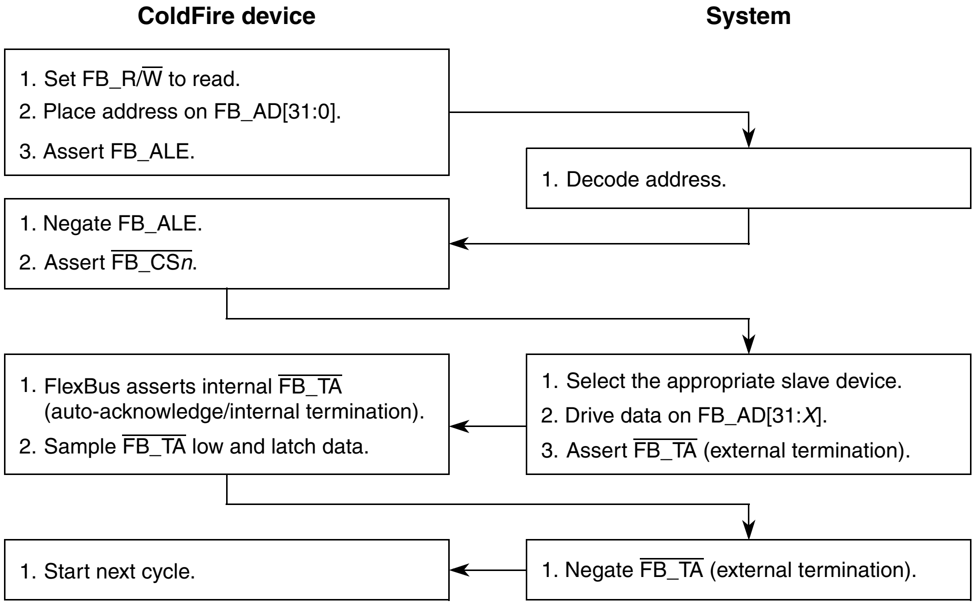

ColdFire device
System
1. Set FB_R/W to read.
2. Place address on FB_AD[31:0].
3. Assert FB_ALE.
1. Decode address.
1. Negate FB_ALE.
2. Assert FB_CSn.
1. FlexBus asserts internal FB_TA (auto-acknowledge/internal termination).
2. Sample FB_TA low and latch data.
1. Select the appropriate slave device.
2. Drive data on FB_AD[31:X].
3. Assert FB_TA (external termination).
1. Start next cycle.
1. Negate FB_TA (external termination).

Figure 20-6. Read Cycle Flowchart

The read cycle timing diagram is shown in Figure 20-7.

NOTE

In the next set of timing diagrams, the dotted lines indicate FB_TA, FB_OE, and FB_CS*n* timing when internal termination is used (CSCR[AA] = 1). The external and internal FB_TA assert at the same time; however, FB_TA is not driven externally for internally-terminated bus cycles.

NOTE

The processor drives the data lines during the first clock cycle of the transfer with the full 32-bit address. This may be ignored by standard connected devices using non-multiplexed address and data buses. However, some applications may find this beneficial.

The address and data busses are muxed between the FlexBus and PCI controller. At the end of the read bus cycles the address signals are indeterminate.

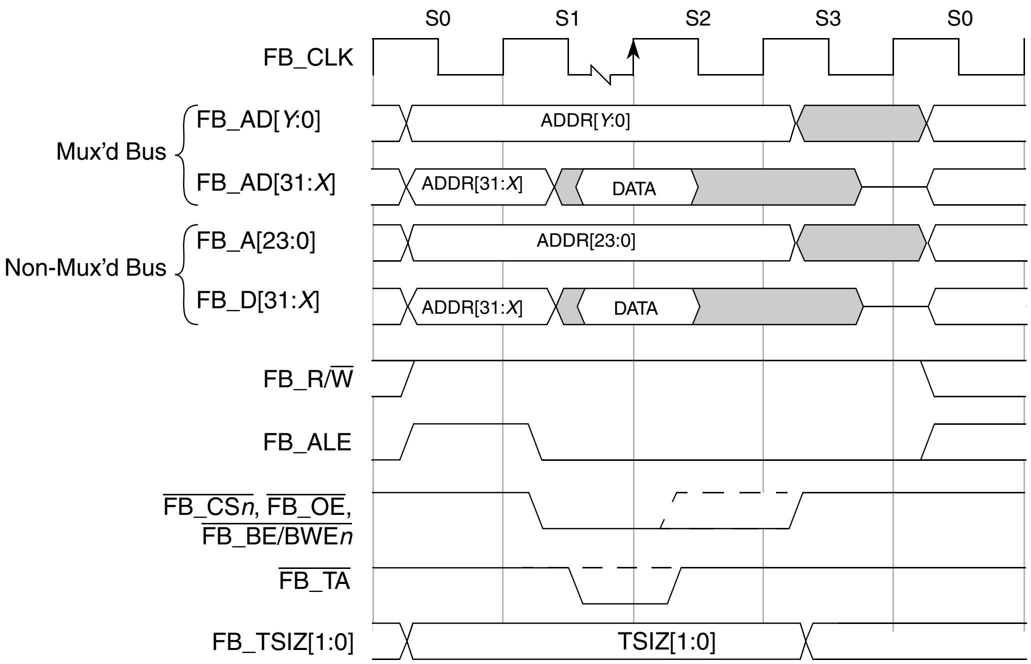


Figure 20-7. Basic Read-Bus Cycle

20.4.6.2 Basic Write Bus Cycle

During a write cycle, the device sends data to memory or to a peripheral device. Figure 20-8 shows the write cycle flowchart.

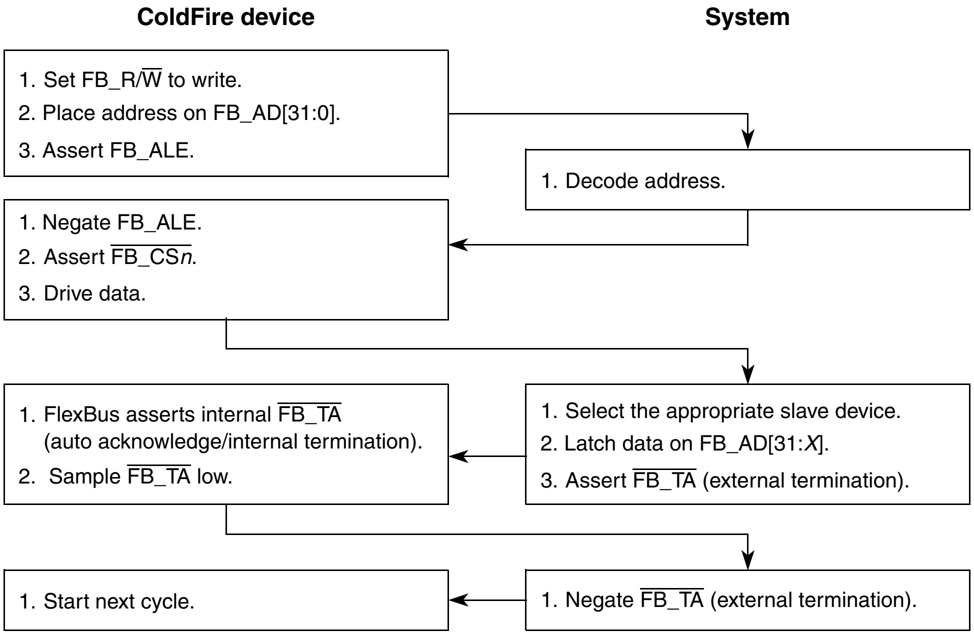

ColdFire device
System
1. Set FB_R/W to write.
2. Place address on FB_AD[31:0].
3. Assert FB_ALE.
1. Decode address.
1. Negate FB_ALE.
2. Assert FB_CSn.
3. Drive data.
1. Select the appropriate slave device.
2. Latch data on FB_AD[31:X].
3. Assert FB_TA (external termination).
1. FlexBus asserts internal FB_TA
(auto acknowledge/internal termination).
2. Sample FB_TA low.
1. Negate FB_TA (external termination).
1. Start next cycle.

Figure 20-8. Write-Cycle Flowchart

Figure 20-9 shows the write cycle timing diagram.

NOTE

The address and data busses are muxed between the FlexBus and PCI controller. At the end of the write bus cycles, the address signals are indeterminate.

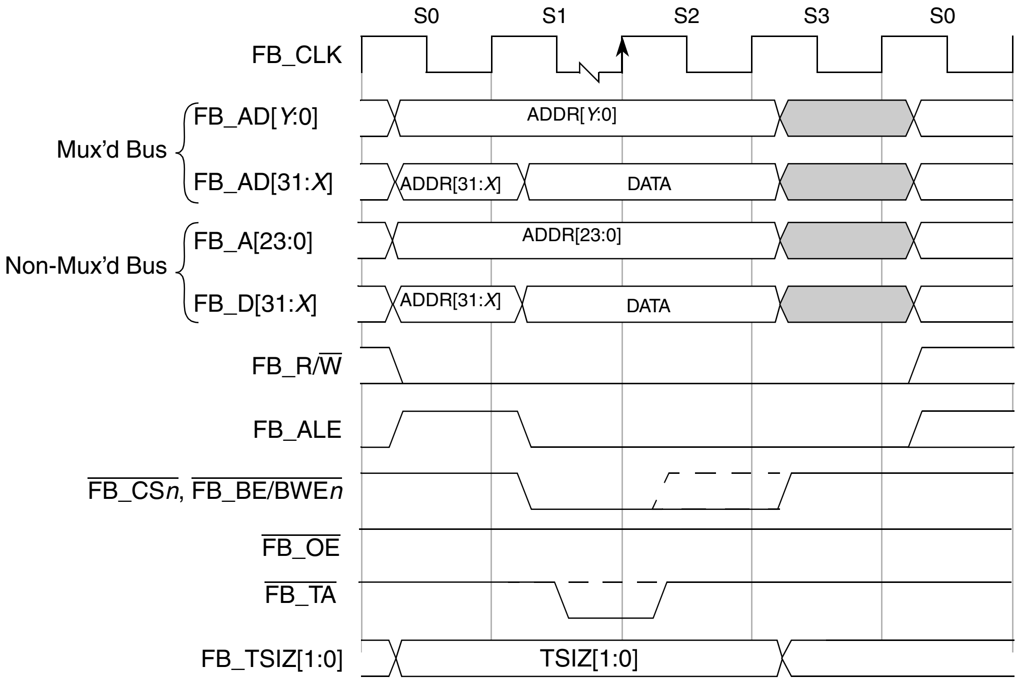


Figure 20-9. Basic Write-Bus Cycle

20.4.6.3 Bus Cycle Sizing

This section shows timing diagrams for various port size scenarios. Figure 20-10 illustrates the basic byte read transfer to an 8-bit device with no wait states. The address is driven on the full FB_AD[31:8] bus in the first clock. The device tristates FB_AD[31:24] on the second clock and continues to drive address on

FB_AD[23:0] throughout the bus cycle. The external device returns the read data on FB_AD[31:24] and may tristate the data line or continue driving the data one clock after FB_TA is sampled asserted.

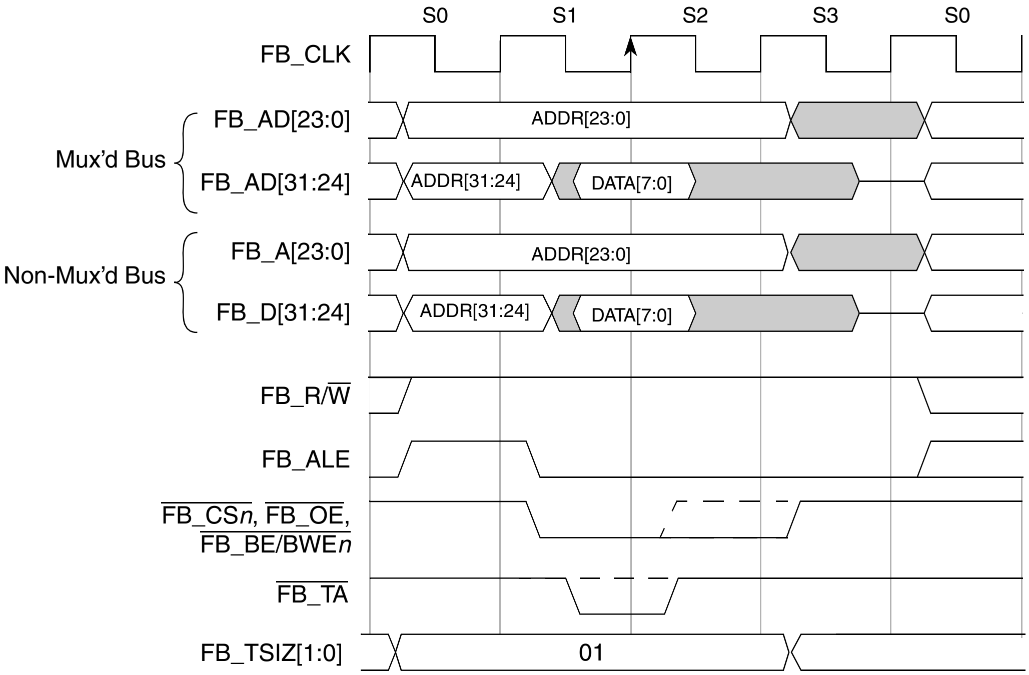


Figure 20-10. Single Byte-Read Transfer

Figure 20-11 shows the similar configuration for a write transfer. The data is driven from the second clock on FB_AD[31:24].

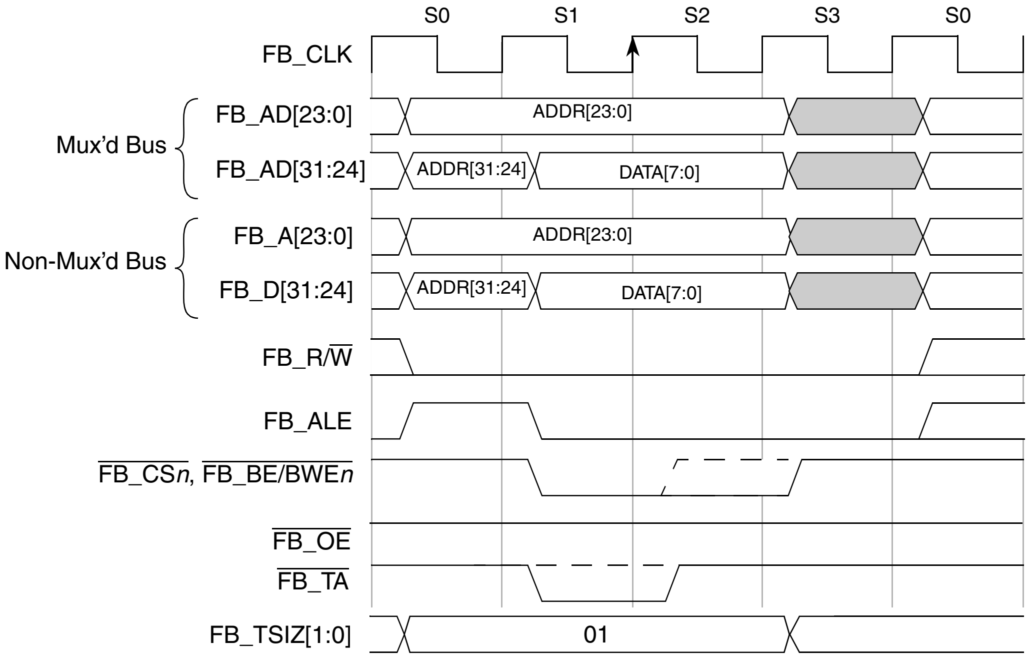


Figure 20-11. Single Byte-Write Transfer

Figure 20-12 illustrates the basic word read transfer to a 16-bit device with no wait states. The address is driven on the full FB_AD[31:8 :0] bus in the first clock. The device tristates FB_AD[31:24] on the second clock and continues to drive the address on FB_AD[15:0] throughout the bus cycle. The external device returns the read data on FB_AD[31:16], and may tristate the data line or continue driving the data one clock after FB_TA is sampled asserted.

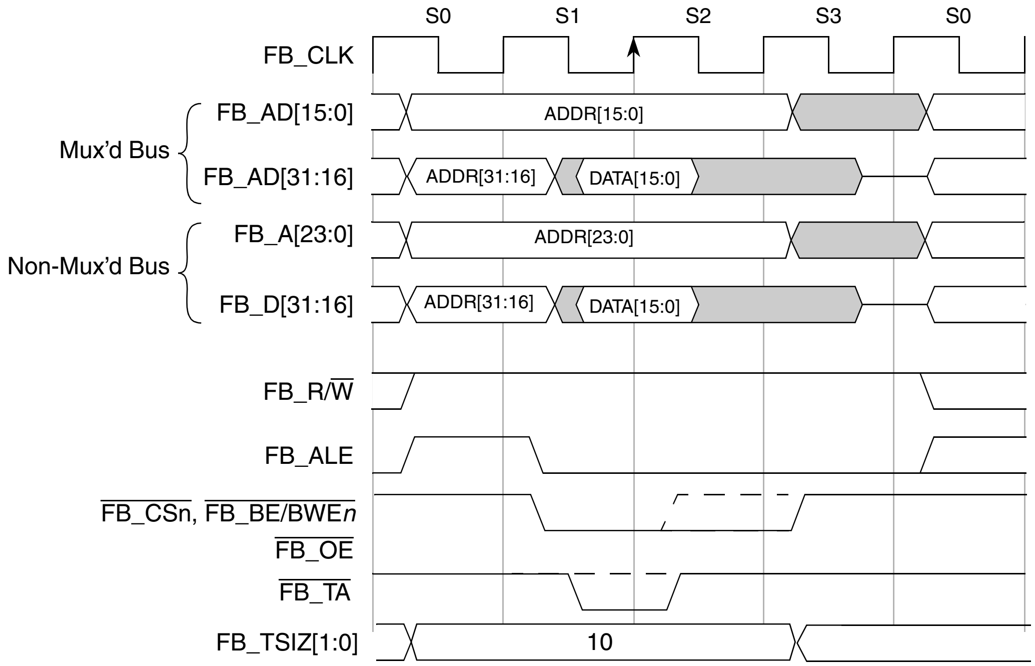


Figure 20-12. Single Word-Read Transfer

Figure 20-13 shows the similar configuration for a write transfer. The data is driven from the second clock on FB_AD[31:16].

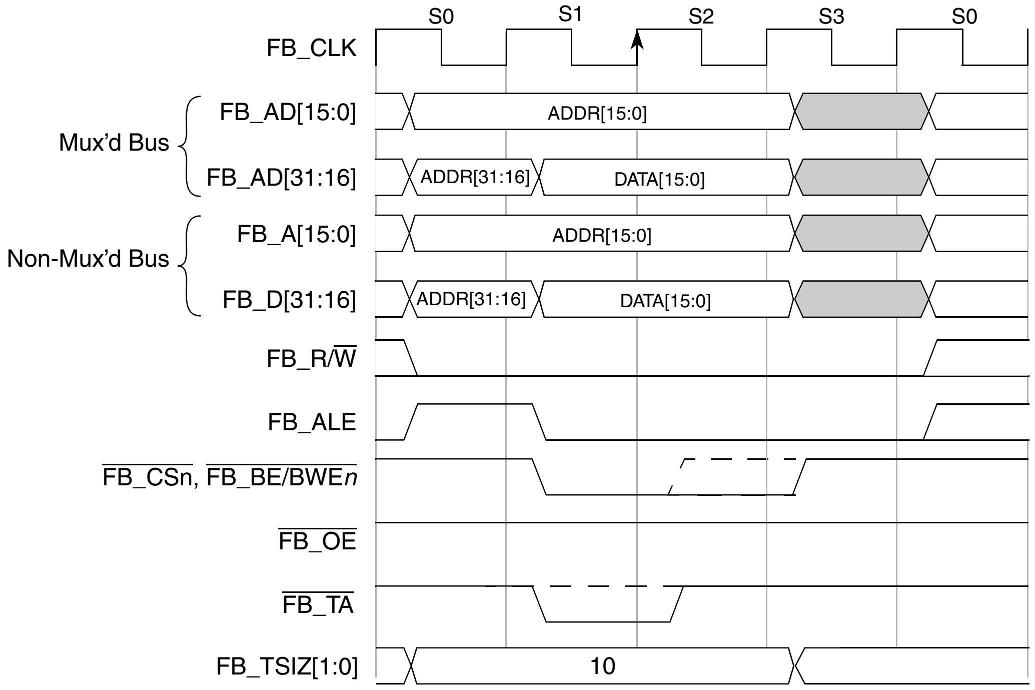


Figure 20-13. Single Word-Write Transfer

Figure 20-14 depicts a longword read from a 32-bit device.

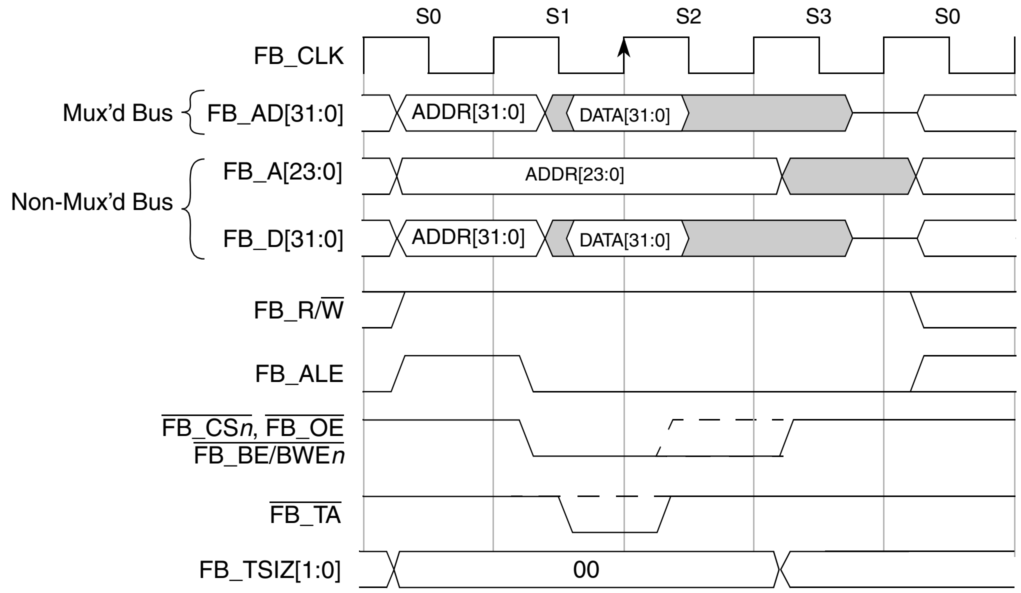


Figure 20-14. Longword-Read Transfer

Figure 20-15 illustrates the longword write to a 32-bit device.

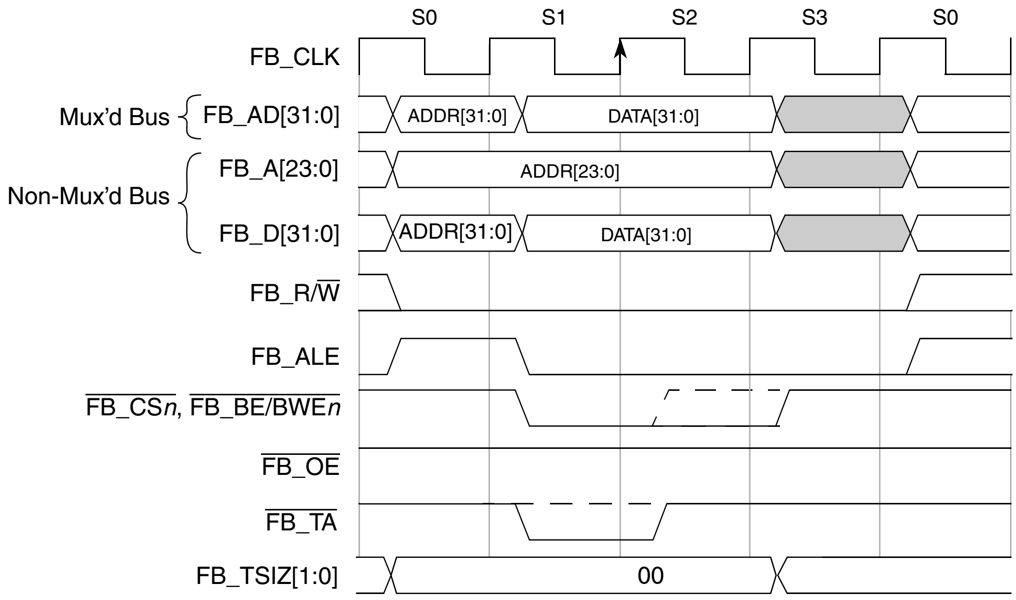


Figure 20-15. Longword-Write Transfer

20.4.6.4 Timing Variations

The FlexBus module has several features that can change the timing characteristics of a basic read- or write-bus cycle to provide additional address setup, address hold, and time for a device to provide or latch data.

20.4.6.4.1 Wait States

Wait states can be inserted before each beat of a transfer by programming the CSCR*n* registers. Wait states can give the peripheral or memory more time to return read data or sample write data.

Figure 20-16 and Figure 20-17 show the basic read and write bus cycles (also shown in Figure 20-7 and Figure 20-12) with the default of no wait states.

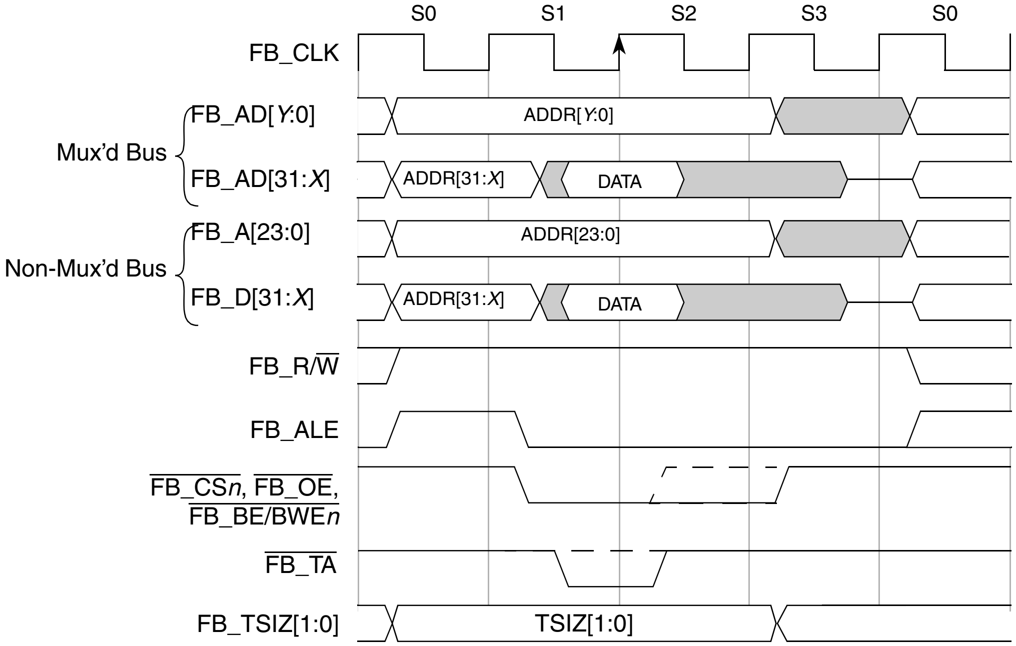


Figure 20-16. Basic Read-Bus Cycle (No Wait States)

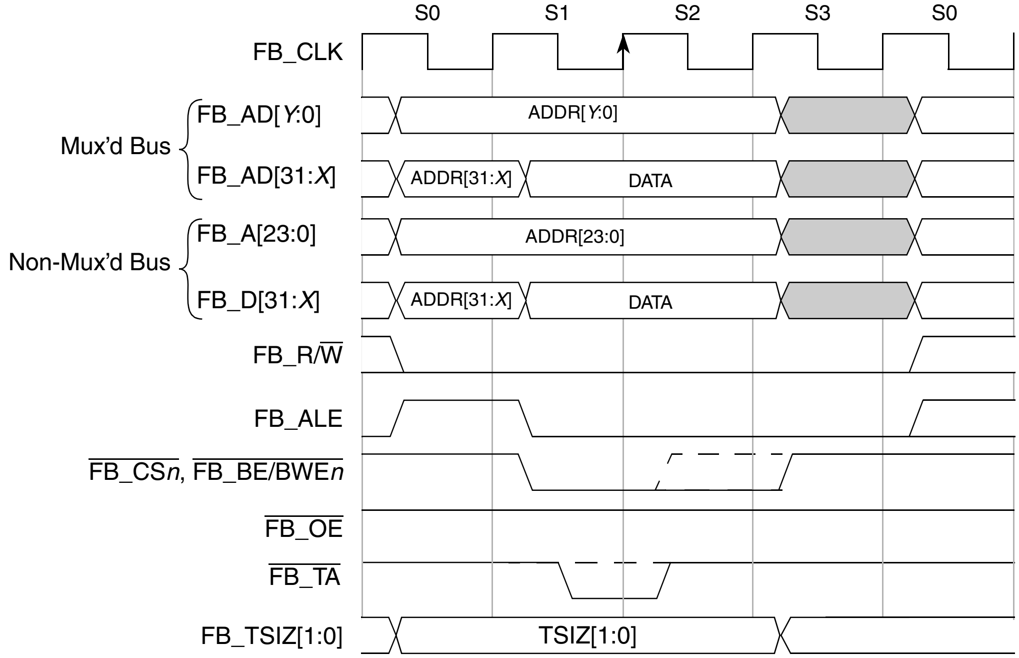


Figure 20-17. Basic Write-Bus Cycle (No Wait States)

If wait states are used, the S1 state repeats continuously until the chip-select auto-acknowledge unit asserts internal transfer acknowledge or the external FB_TA is recognized as asserted. Figure 20-18 and Figure 20-19 show a read and write cycle with one wait state.

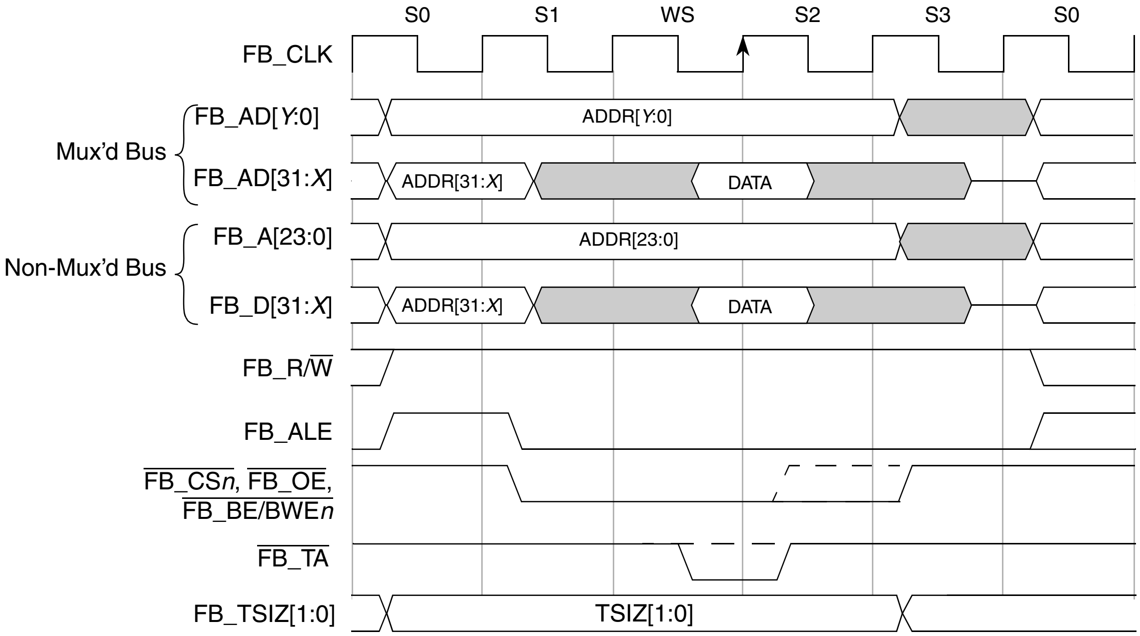


Figure 20-18. Read-Bus Cycle (One Wait State)

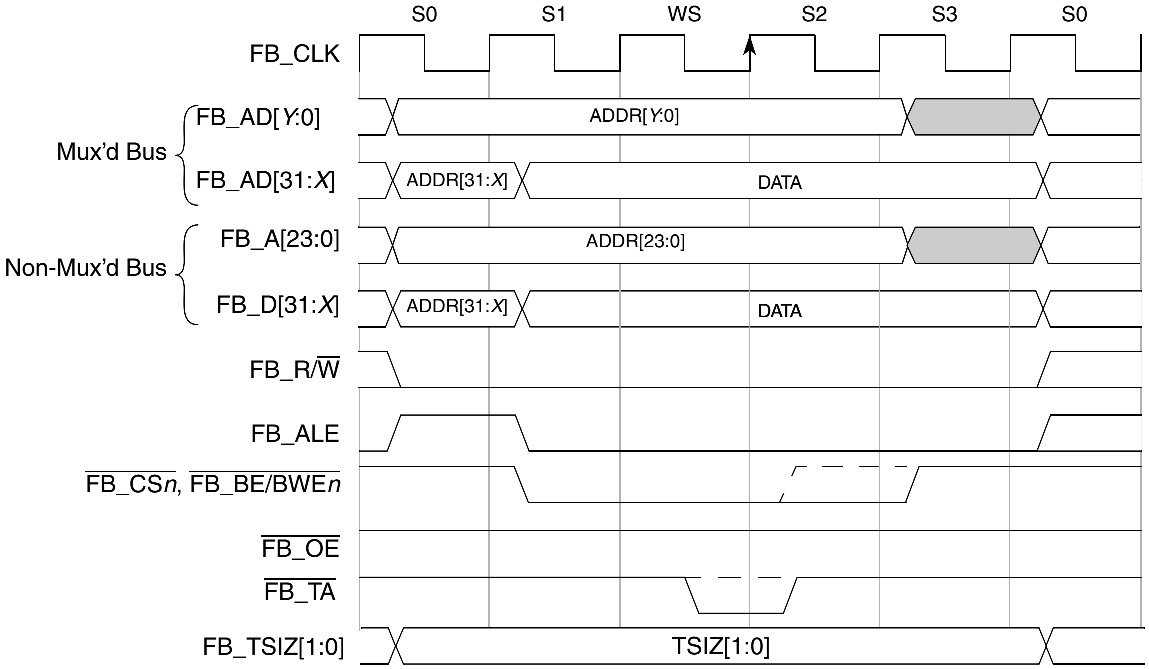


Figure 20-19. Write-Bus Cycle (One Wait State)

20.4.6.4.2 Address Setup and Hold

The timing of the assertion and negation of the chip selects, byte selects, and output enable can be programmed on a chip-select basis. Each chip-select can be programmed to assert one to four clocks after address-latch enable (FB_ALE) is asserted. Figure 20-20 and Figure 20-21 show read- and write-bus cycles with two clocks of address setup.

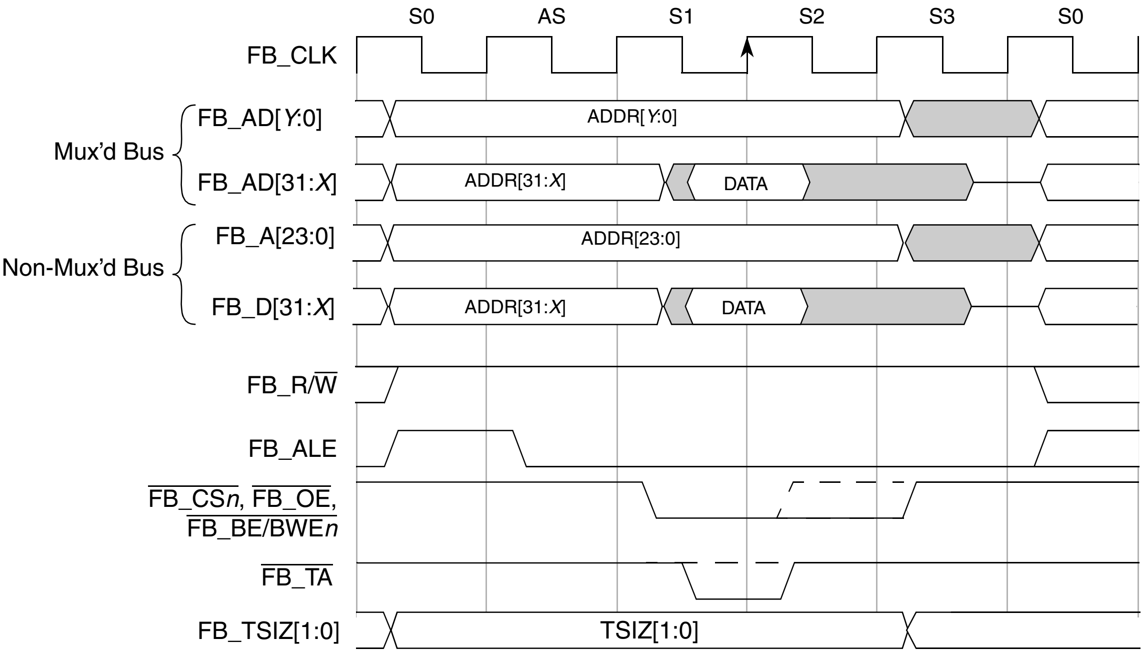


Figure 20-20. Read-Bus Cycle with Two-Clock Address Setup (No Wait States)

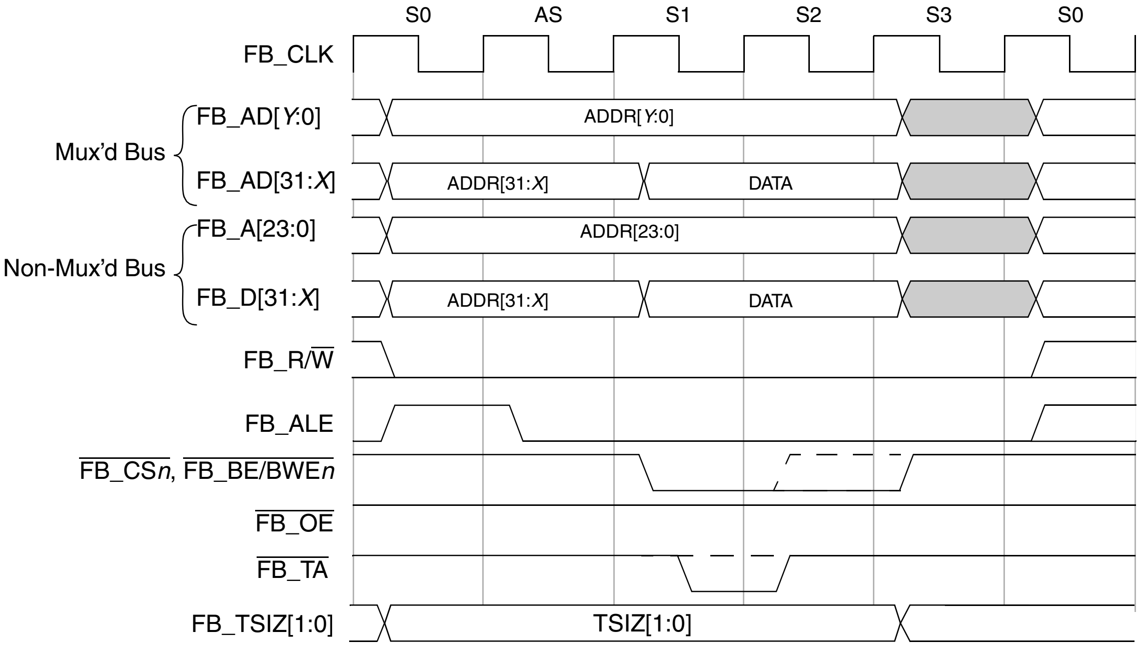


Figure 20-21. Write-Bus Cycle with Two Clock Address Setup (No Wait States)

In addition to address setup, a programmable address hold option for each chip select exists. Address and attributes can be held one to four clocks after chip-select, byte-selects, and output-enable negate. Figure 20-22 and Figure 20-23 show read and write bus cycles with two clocks of address hold.

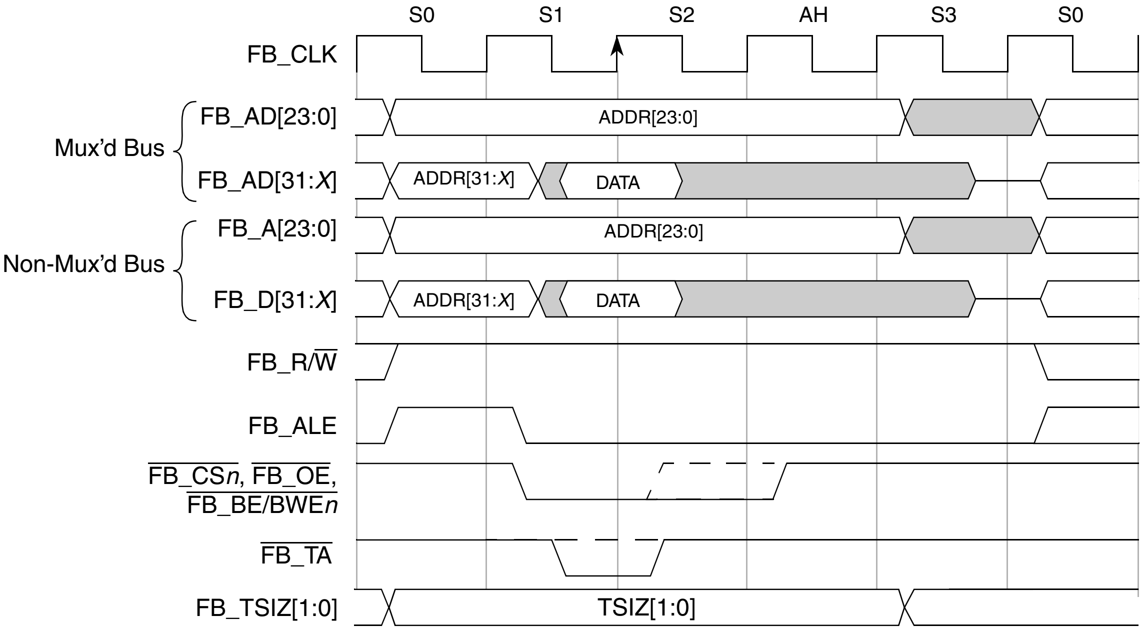


Figure 20-22. Read Cycle with Two-Clock Address Hold (No Wait States)

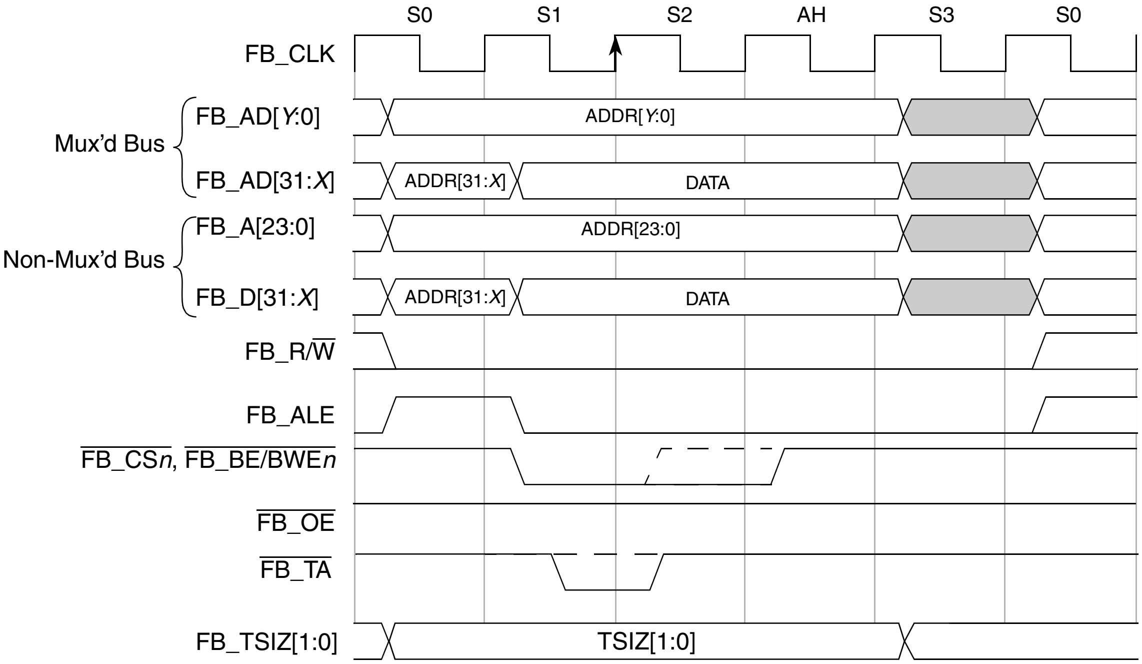


Figure 20-23. Write Cycle with Two-Clock Address Hold (No Wait States)

Figure 20-24 shows a bus cycle using address setup, wait states, and address hold.

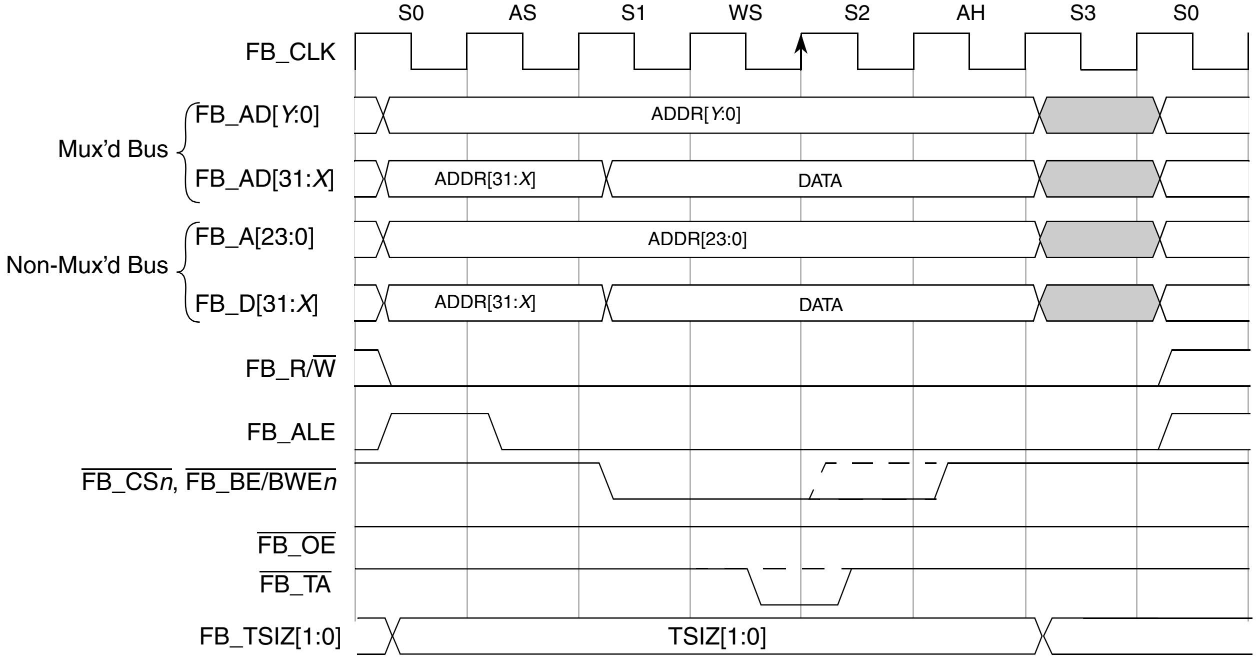


Figure 20-24. Write Cycle with Two-Clock Address Setup and Two-Clock Hold (One Wait State)

20.4.7 Burst Cycles

The device can be programmed to initiate burst cycles if its transfer size exceeds the port size of the selected destination. The initiation of a burst cycle is encoded on the size pins. For burst transfers to smaller port sizes, FB_TSIZ[1:0] indicates the size of the entire transfer. For example, with bursting enabled, a word transfer to an 8-bit port takes two beats (two byte-sized transfers), for which FB_TSIZ[1:0] equals 10 throughout. A longword transfer to an 8-bit port would take a 4-byte burst cycle, for which FB_TSIZ[1:0] equals 00 throughout.

With bursting disabled, any transfer larger than the port size breaks into multiple individual transfers. With bursting enabled, an access larger than port size results in a burst cycle of multiple beats. Table 20-11 shows the result of such transfer translations.

Table 20-11. Transfer Size and Port Size Translation

Port Size PS[1:0]	Transfer Size FB_TSIZ[1:0]	Burst-Inhibited: Number of Transfers Burst Enabled: Number of Beats
01 (8-bit)	10 (word)	2
	00 (longword)	4
	11 (line)	16
1*x* (16-bit)	00 (longword)	2
	11 (line)	8
00 (32-bit)	11 (line)	4

The FlexBus can support 2-1-1-1 burst cycles to maximize system performance. Delaying termination of the cycle can add wait states. If internal termination is used, different wait state counters can be used for the first access and the following beats.

The CSCR*n* registers enable bursting for reads, writes, or both. Memory spaces can be declared burst-inhibited for reads and writes by clearing the appropriate CSCR*n*[BSTR,BSTW] bits.

Figure 20-25 shows a longword read to an 8-bit device programmed for burst enable. The transfer results in a 4-beat burst and the data is driven on FB_AD[31:24]. The transfer size is driven at longword (00) throughout the bus cycle.

NOTE

In non-multiplexed address/data mode, the address on FB_A increments only during internally-terminated burst cycles. The first address is driven throughout the entire burst for externally-terminated cycles.

In multiplexed address/data mode, the address is driven on FB_AD only during the first cycle for internally- and externally-terminated cycles.

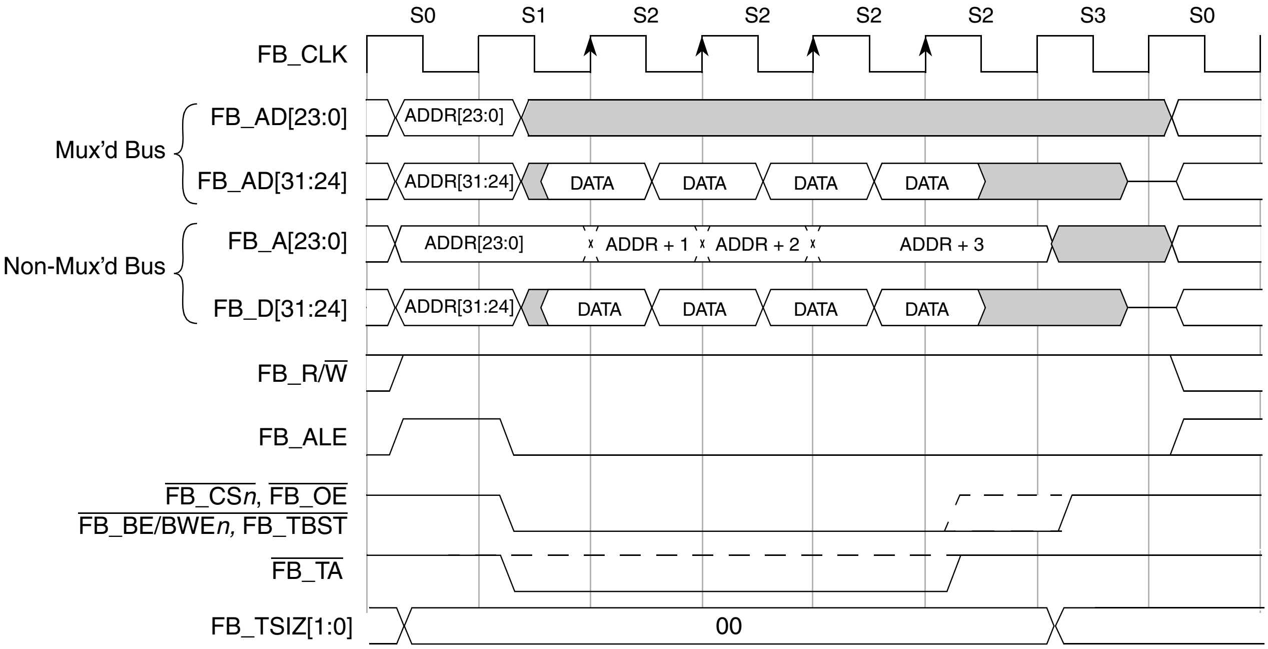


Figure 20-25. Longword-Read Burst from 8-Bit Port 2-1-1-1 (No Wait States)

Figure 20-26 shows a longword write to an 8-bit device with burst enabled. The transfer results in a 4-beat burst and the data is driven on FB_AD[31:24]. The transfer size is driven at longword (00) throughout the bus cycle.

NOTE

The first beat of any write burst cycle has at least one wait state. If the bus cycle is programmed for zero wait states (CSCR*n*[WS] = 0), one wait state is added. Otherwise, the programmed number of wait states are used.

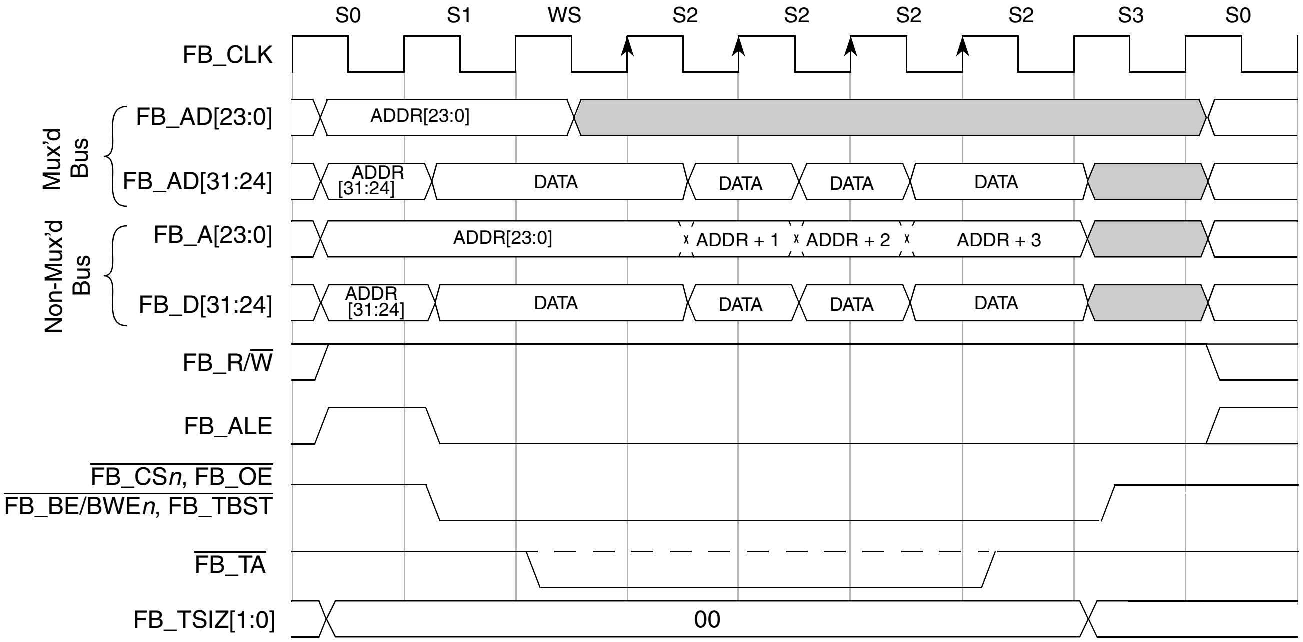


Figure 20-26. Longword-Write Burst to 8-Bit Port 3-1-1-1 (No Wait States)

Figure 20-27 shows a longword read from an 8-bit device with burst inhibited. The transfer results in four individual transfers. The transfer size is driven at longword (00) during the first transfer and at byte (01) during the next three transfers.

NOTE

There is an extra clock of address setup (AS) for each burst-inhibited transfer between states S0 and S1.

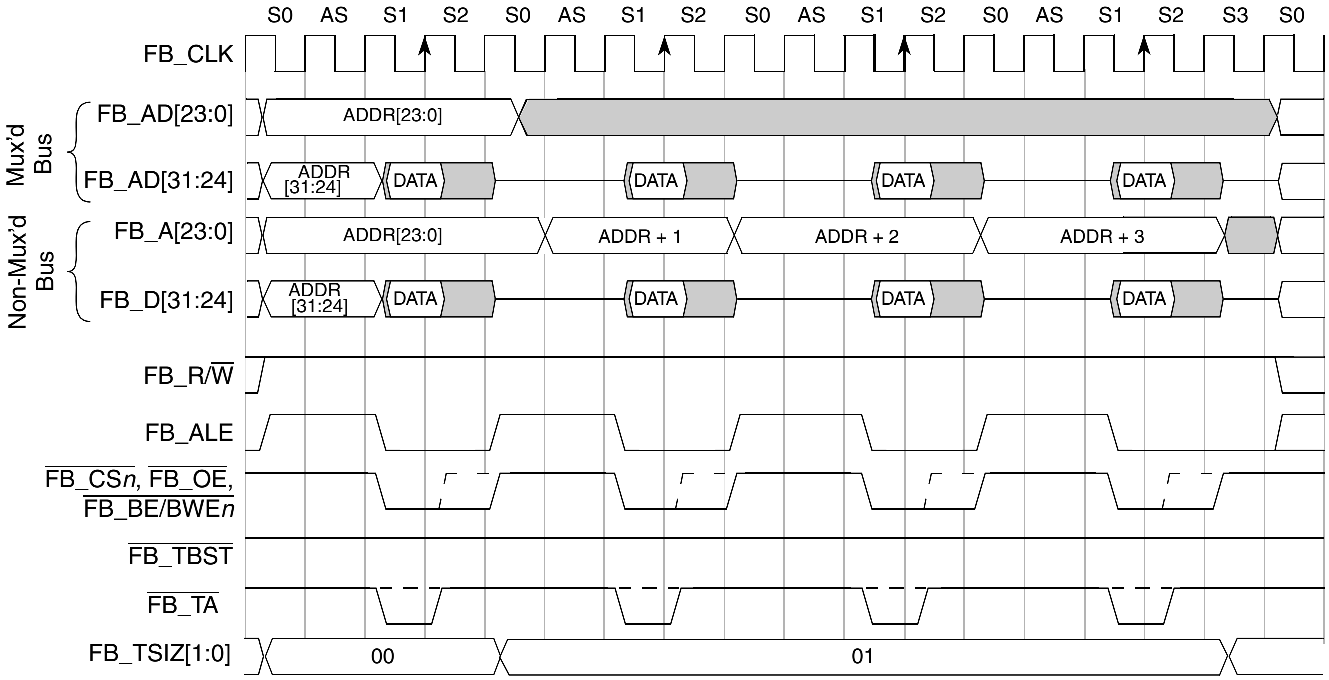


Figure 20-27. Longword-Read Burst-Inhibited from 8-Bit Port (No Wait States)

Figure 20-28 shows a longword write to an 8-bit device with burst inhibited. The transfer results in four individual transfers. The transfer size is driven at longword (00) during the first transfer and at byte (01) during the next three transfers.

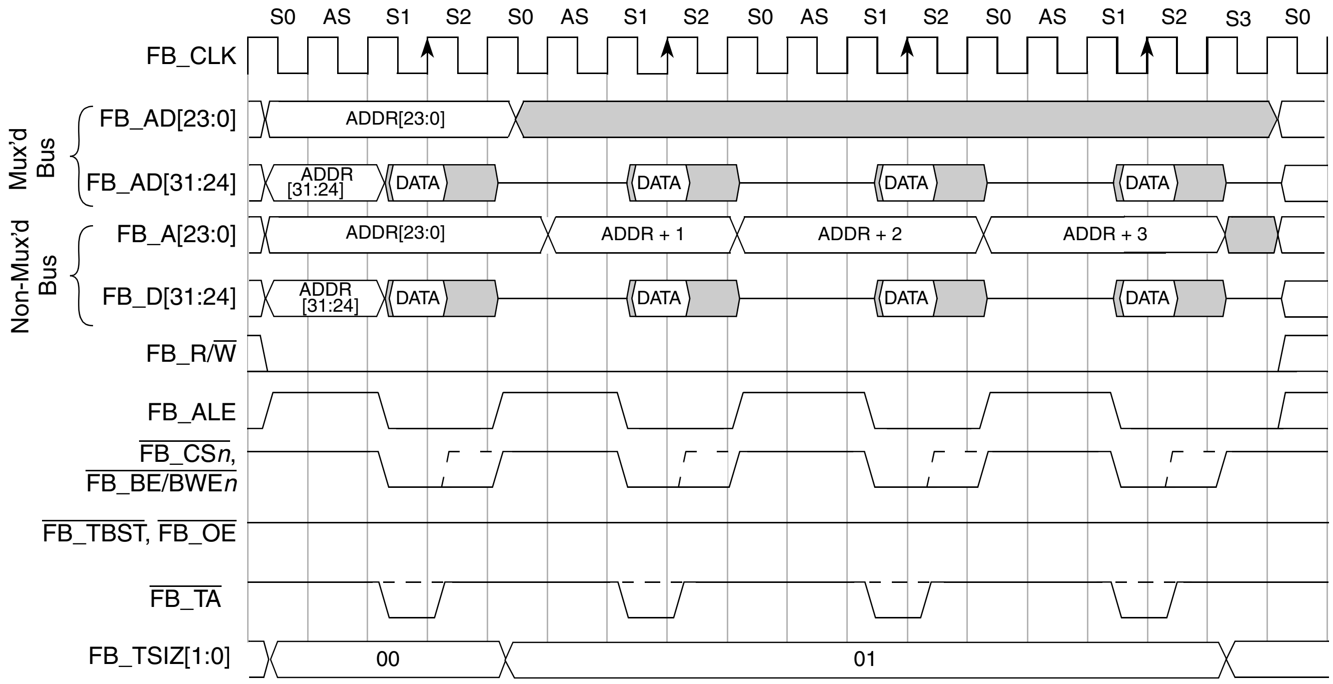


Figure 20-28. Longword-Write Burst-Inhibited to 8-Bit Port (No Wait States)

Figure 20-29 illustrates another read burst transfer, but in this case a wait state is added between individual beats.

NOTE

CSCR*n*[WS] determines the number of wait states in the first beat. However, for subsequent beats, the CSCR*n*[WS] (or CSCR*n*[SWS] if CSCR*n*[SWSEN] is set) determines the number of wait states.

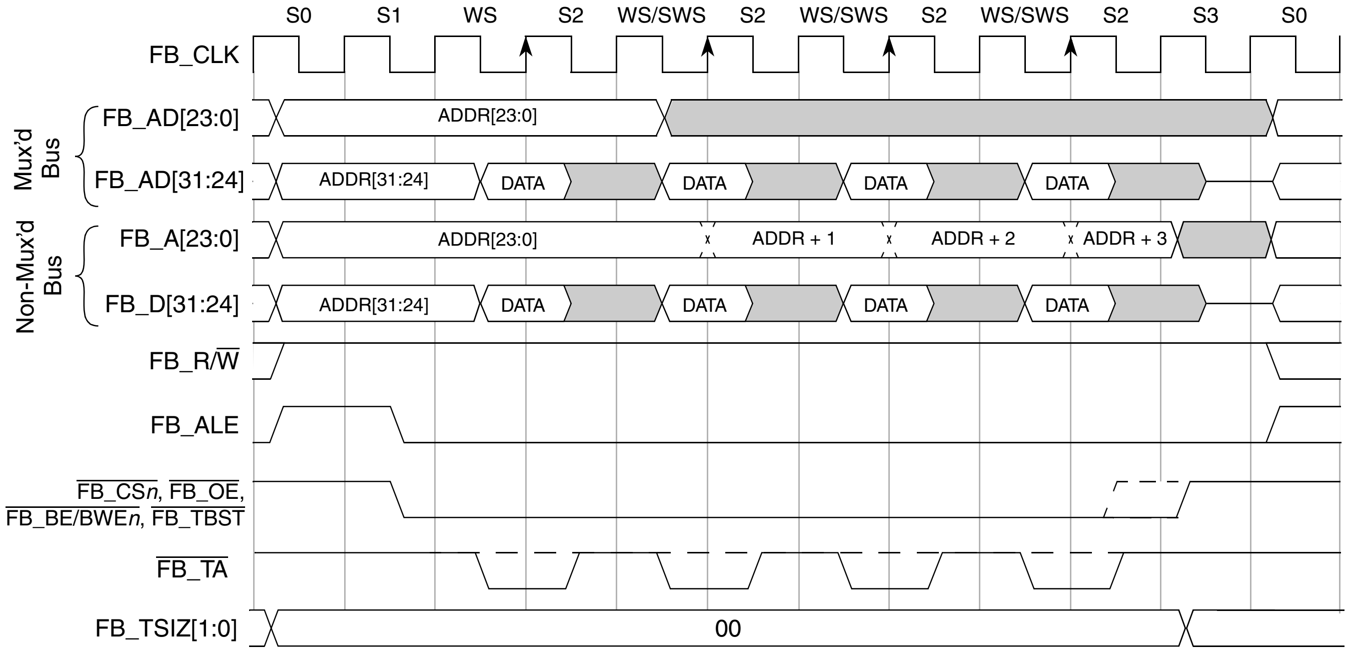


Figure 20-29. Longword-Read Burst from 8-Bit Port 3-2-2-2 (One Wait State)

Figure 20-29 illustrates a write burst transfer with one wait state.

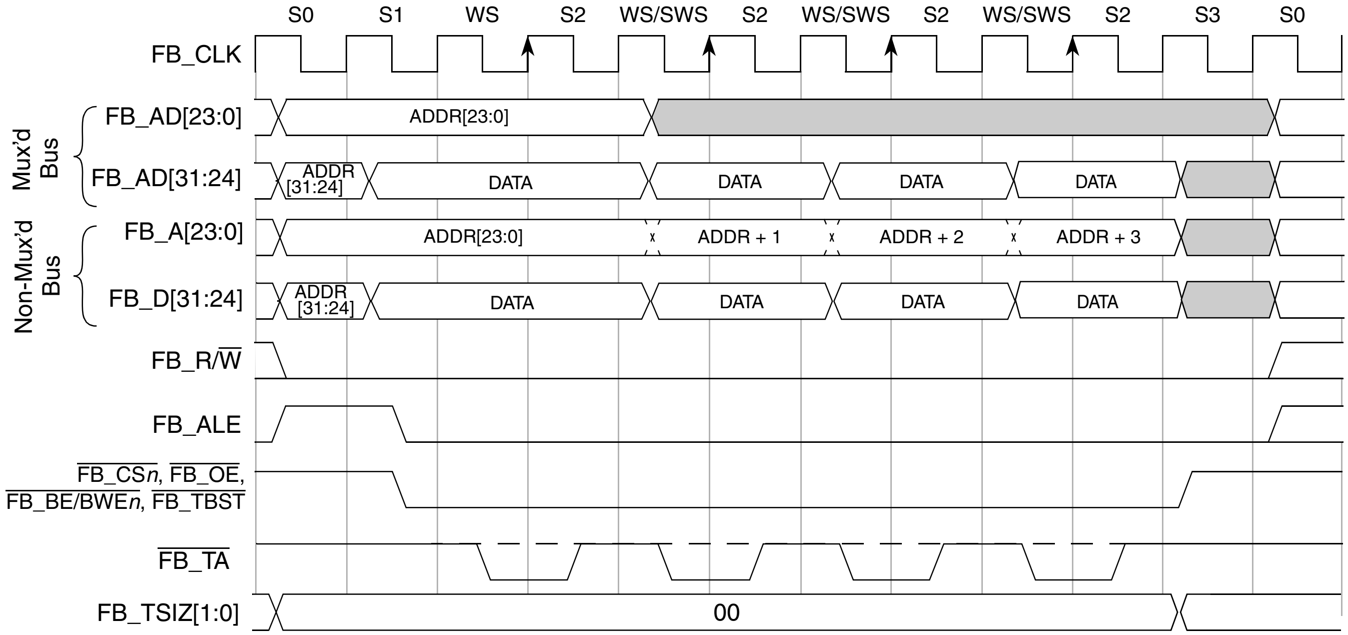
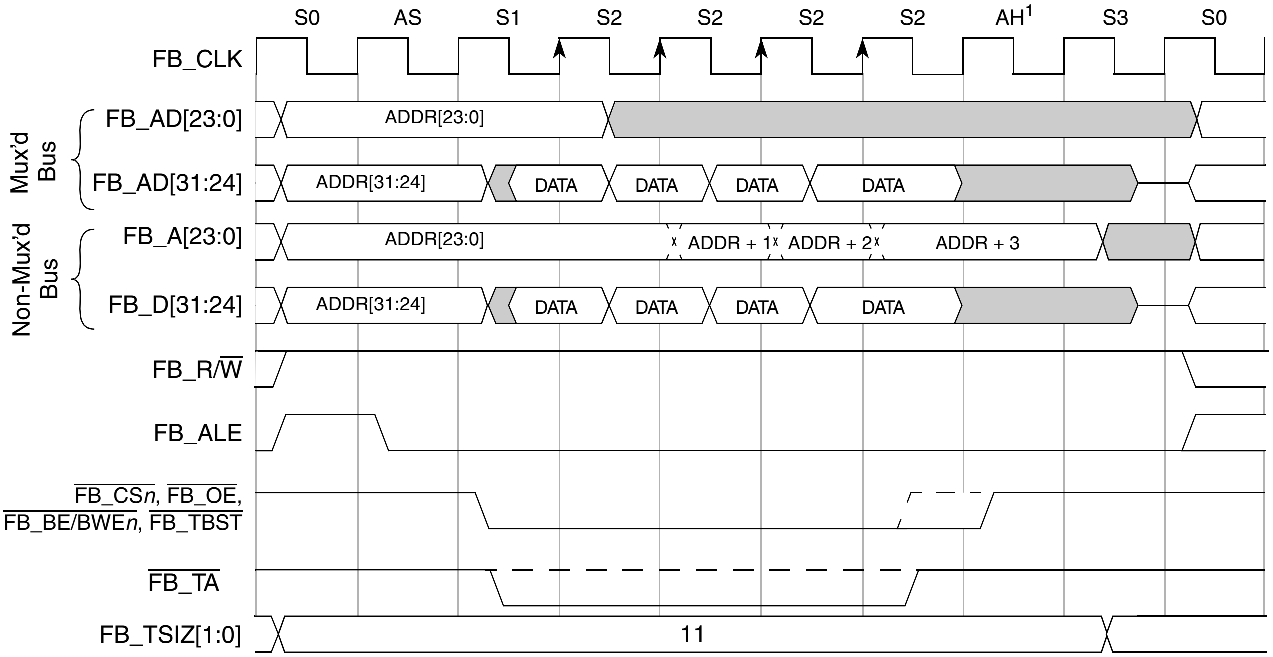


Figure 20-30. Longword-Write Burst to 8-Bit Port 3-2-2-2 (One Wait State)

If address setup and hold are used, only the first and last beat of the burst cycle are affected. Figure 20-31 shows a read cycle with one clock of address setup and address hold.

NOTE

> In non-multiplexed address/data mode, the address on FB_A increments only during internally-terminated burst cycles (CSCR*n*[AA] = 1). The attached device must be able to account for this, or a wait state must be added. The first address is driven throughout the entire burst for externally-terminated cycles.
>
> In multiplexed address/data mode, the address is driven on FB_AD only during the first cycle for internally- and externally-terminated cycles.



[1] The address hold time depends on the setting of CSCR*n*[AA]. See Section 20.3.3, "Chip-Select Control Registers (CSCR0 – CSCR5)", for more details.

Figure 20-31. Longword-Read Burst from 8-Bit Port 3-1-1-1 (Address Setup and Hold)

Figure 20-32 shows a write cycle with one clock of address setup and address hold.

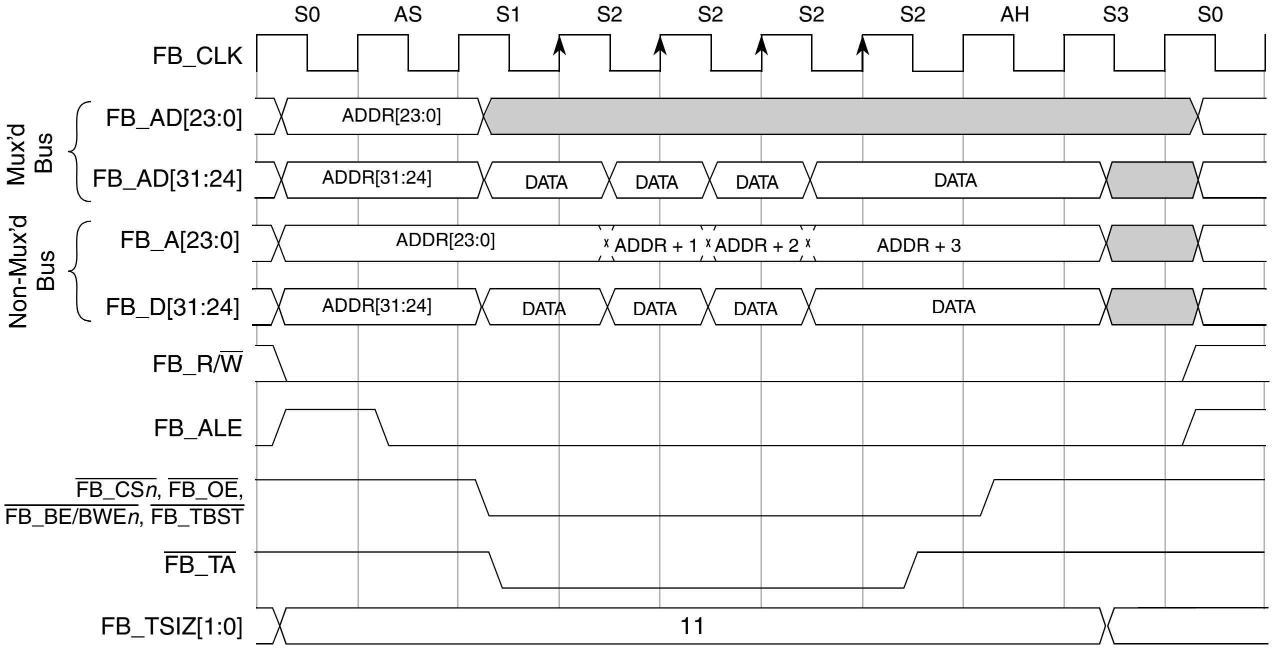


Figure 20-32. Longword-Write Burst to 8-Bit Port 3-1-1-1 (Address Setup and Hold)

20.4.8 Misaligned Operands

Because operands, unlike opcodes, can reside at any byte boundary, they are allowed to be misaligned.

- Byte operand is properly aligned at any address
- Word operand is misaligned at an odd address
- Longword is misaligned at any address not a multiple of four

Although the processor enforces no alignment restrictions for data operands (including program counter (PC) relative data addressing), misaligned operands require additional bus cycles.

Instruction words and extension words (opcodes) must reside on word boundaries. Attempting to prefetch a misaligned instruction word causes an address-error exception.

The processor core converts misaligned, cache-inhibited operand accesses to multiple aligned accesses. Example 20-1 shows the transfer of a longword operand from a byte address to a 32-bit port. First, a byte transfers at an offset of 0x1. The slave device supplies the byte and acknowledges the data transfer. When the processor starts the second cycle, a word transfers with a byte offset of 0x2. The next two bytes are transferred in this cycle. In the third cycle, byte 3 transfers. The byte offset is now 0x0, the port supplies the final byte, and the operation completes.

	31 24	23 16	15 8	7 0	FB_A[2:0]
Transfer 1	—	Byte 0	—	—	001
Transfer 2	—	—	Byte 1	Byte 2	010
Transfer 3	Byte 3	—	—	—	100

Example 20-1. A Misaligned Longword Transfer (32-Bit Port)

If an operand is cacheable and is misaligned across a cache-line boundary, both lines are loaded into the cache. The example in Example 20-2 differs from the one in Example 20-1 because the operand is word-sized and the transfer takes only two bus cycles.

	31 24	23 16	15 8	7 0	FB_A[2:0]
Transfer 1	—	—	—	Byte 0	001
Transfer 2	Byte 0	—	—	—	100

Example 20-2. A Misaligned Word Transfer (32-Bit Port)

20.4.9 Bus Errors

If the auto-acknowledge is not enabled for the address that generates the error, the bus cycle can be terminated by asserting FB_TA. If the processor must manage a bus error differently, asserting an interrupt to the core along with FB_TA when the bus error occurs can invoke an interrupt handler.

The device also includes a bus monitor that generates a bus error for unterminated cycles.

Chapter 21
SDRAM Controller (SDRAMC)

21.1 Introduction

This chapter describes configuration and operation of the synchronous DRAM (SDRAM) controller. It begins with a general description and brief glossary and includes a description of signals involved in DRAM operations. The remainder of the chapter describes the programming model and signal timing, as well as the command set required for synchronous operations. It also includes examples to better understand how to configure the DRAM controller for synchronous operations.

NOTE

Unless otherwise noted, in this chapter clock refers to the system clock ($f_{sys/2}$).

21.1.1 Block Diagram

Block diagram of the SDRAM controller:

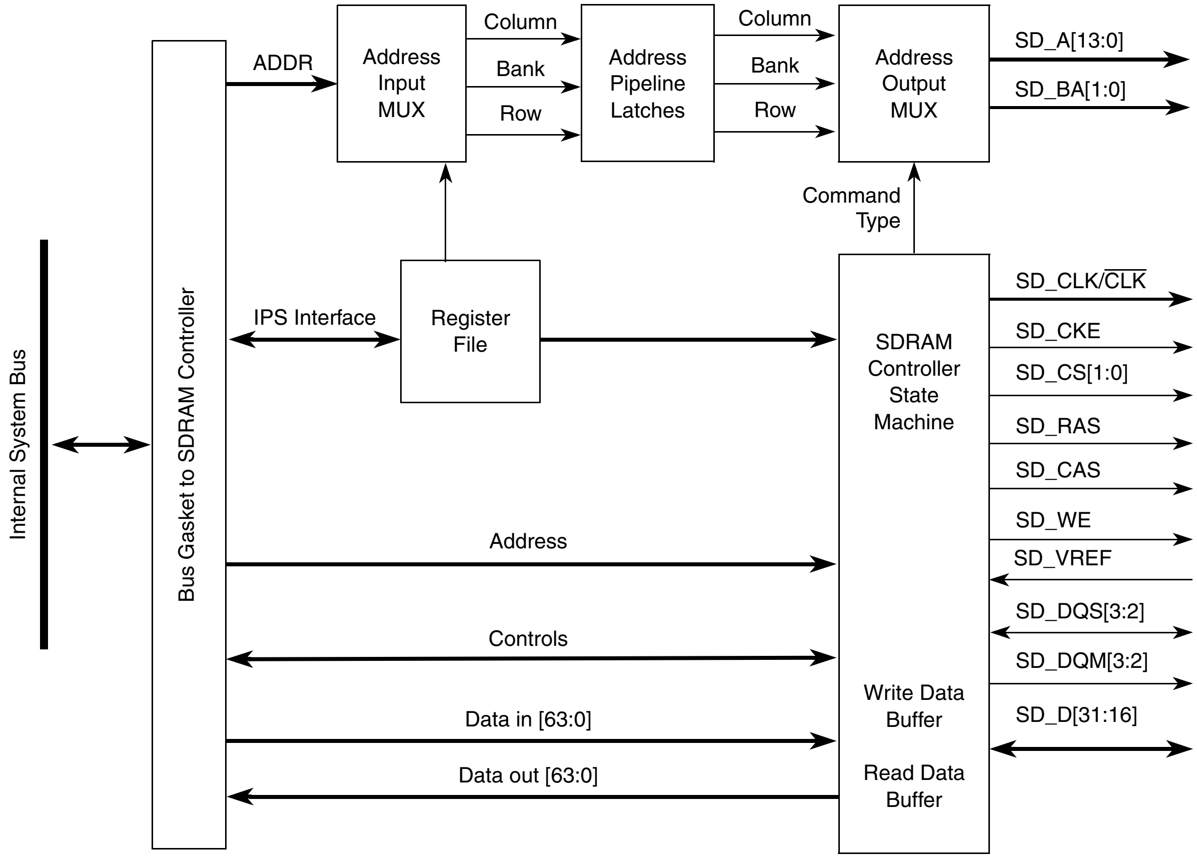


Figure 21-1. SDRAM Controller Block Diagram

21.1.2 Features

The SDRAM controller contains:

- Supports dual data rate (DDR) SDRAM
- Support for lower-power/mobile DDR SDRAM
- Support for DDR2 SDRAM.
- 16-bit fixed memory data port width
- 16 bytes critical word first burst transfer. Supports sequential address order only
- Up to 14 lines of row address, up to 13 (in 16-bit bus mode) column address lines, 2 bits of bank address, and two pinned-out chip selects. The maximum row bits plus column bits equals 25 in 16-bit bus mode.
- Minimum memory configuration of 8 MByte
  - — 11 bit row address (RA), 9 bit column address (CA), 2 bit bank address (BA), 16-bit bus, one chip select

- Supports up to 512 MByte of memory.
  - — 25 bits RA+CA, 2 bits BA, 16-bit bus, two chip selects
- Supports page mode for decreased latency and higher bandwidth; remembers one active row for each bank; four independent active rows per each chip select
- Programmable refresh interval timer
- Supports sleep mode and self-refresh mode
- Error detect and parity check are not supported
- The SDRAM controller does not include a dedicated I^{2}C interface to access memory module (DIMM) serial presence detect EEPROM. If needed, this must be managed by one of the on-chip I^{2}C channels external to the SDRAM controller.
- Read clock recovery block

21.1.3 Terminology

The following terminology is used in this chapter:

- SDRAM block: Any group of DRAM memories selected by one of the $\overline{\text{SD\_CS}}$ signals. Therefore, the SDRAMC can support up to two independent memory blocks. The base address of each block is programmed in the SDRAM chip-select configuration registers.
- SDRAM bank: An internal partition in an SDRAM device. For example, a 64-Mbit SDRAM component might be configured as four 512K x 32 banks. Banks are selected through the SD_BA[1:0] signals.
- SDRAM: RAMs that operate like asynchronous DRAMs but with a synchronous clock, a pipelined, multiple-bank architecture, and a faster speed.

21.2 External Signal Description

This section introduces the signal names used in this chapter.

Table 21-1. SDRAM Interface—Detailed Signal Descriptions

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="2">SD_A[13:0]</td><td rowspan="2">O</td><td colspan="2">Memory multiplexed row/column address. Provides the row address for ACTV commands, and the column address and auto-precharge bit for READ/WRITE commands, to select one location out of the memory array in the respective bank. A10 is sampled during a precharge command to determine whether the precharge applies to one bank (A10 negated) or all banks (A10 asserted). If only one bank is to be precharged, the bank is selected by SD_BA[1:0].
The address outputs also provide the opcode during a MODE REGISTER SET command. SD_BA[1:0] signals define which mode register is loaded during the MODE REGISTER SET (MRS or EMRS). A12 is used on device densities of 256 Mb and above.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation — Occurs synchronously with SD_CLK</td></tr>
<tr><td rowspan="2">SD_BA[1:0]</td><td rowspan="2">O</td><td colspan="2">Memory bank address. Define which bank an ACTV, READ, WRITE, or PRECHARGE command is being applied. It is also used to select the SDRAM internal mode register during power-up initialization.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation — Occurs synchronously with SD_CLK</td></tr>
</table>

Table 21-1. SDRAM Interface—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{SD\_CAS}}$	O	Column address strobe/command input. Along with $\overline{\text{SD\_CS}}$, $\overline{\text{SD\_RAS}}$, and $\overline{\text{SD\_WE}}$, defines the current command.	
		State Meaning	See Table 21-10 for the SDRAM commands.
		Timing	Assertion/Negation — Occurs synchronously with SD_CLK
$\overline{\text{SD\_RAS}}$	O	Row address strobe/command input. Along with $\overline{\text{SD\_CS}}$, $\overline{\text{SD\_CAS}}$, and $\overline{\text{SD\_WE}}$, defines the current command.	
		State Meaning	See Table 21-10 for SDRAM commands.
		Timing	Assertion/Negation — Occurs synchronously with SD_CLK.
SD_CKE	O	Clock enable. SD_CKE must be maintained high throughout READ and WRITE accesses. SD_CKE negates to put the SDRAM into low-power, self-refresh mode. Input buffers, excluding SD_CLK, $\overline{\text{SD\_CLK}}$, and SD_CKE, are disabled during self-refresh.	
		State Meaning	Asserted — Activates internal clock signals and device input buffers and output drivers. Negated —Deactivates internal clock signals and device input buffers and output drivers.
		Timing	Assertion — Asynchronous for self-refresh exit and for output disable Negation — Occurs synchronously with SD_CLK
SD_CLK $\overline{\text{SD\_CLK}}$	O	SD_CLK and $\overline{\text{SD\_CLK}}$ are differential clock outputs. All address and control output signals are sent on the crossing of the positive edge of SD_CLK and the negative edge of $\overline{\text{SD\_CLK}}$. Output data is referenced to the crossing of SD_CLK and $\overline{\text{SD\_CLK}}$ (both directions of crossing).	
		Timing	Command signals occur synchronously with the rising edge of this clock. Data signals can change on the rising and falling edge of the clock.
$\overline{\text{SD\_CS}}$[1:0]	O	$\overline{\text{SD\_CS}}$ provides external bank selection on systems with multiple banks. $\overline{\text{SD\_CS}}$ is considered part of the command code.	
		State Meaning	Asserted — Commands for the selected chip occur Negated — All commands are masked.
		Timing	Assertion/Negation — Occurs synchronously with SD_CLK
SD_DATA[31:16]	I/O	Data bus. In 16-bit DDR configuration, the memory device data bus is connected to SD_D[31:16] bits.	
		Timing	Assertion/Negation — Occurs on crossing of SD_CLK and $\overline{\text{SD\_CLK}}$. High Impedance - Depending on the OE_RULE bit in SDCFG1, the SD_DATA bus can be in high impedance until a write occurs or only when a read occurs.
$\overline{\text{SD\_DQM}}$[3:2]	O	Output mask signal for write data. During reads, $\overline{\text{SD\_DQM}}$ may be driven high, low, or floating. The address correspondence: SD_DQM3 - SD_D[31:24] SD_DQM2 - SD_D[23:16]	
		State Meaning	Asserted — Data is written to SDRAM Negation — Data is masked
		Timing	Assertion/Negation — Occurs on crossing of SD_CLK and $\overline{\text{SD\_CLK}}$.

Table 21-1. SDRAM Interface—Detailed Signal Descriptions (continued)

<table>
<tr><th>Signal</th><th>I/O</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">SD_DQS[3:2]</td><td rowspan="3">I/O</td><td colspan="2">Data strobes that indicate valid read/write data. (Edge-aligned with read data, centered with write data.) The DQS frequency equals the memory clock frequency. Data is normally 1/4 memory clock period after a DQS transition. For DDR operation, there is data following each DQS edge (rising and falling); for SDR operation, valid data follows the rising edges only. The address correspondence:
SD_DQS3 - SD_D[31:24]
SD_DQS2 - SD_D[23:16]
<b>Note:</b> If a read is attempted from a DDR SDRAM chip select when there is no memory to respond with the appropriate SD_DQS pulses, the bus cycle hangs. Because there is no high level bus monitor on the device, a reset is the only way to exit this error condition.</td></tr>
<tr><td><b>State Meaning</b></td><td>Asserted — Similar to a clock signal, the edges are more important than being asserted or negated.
High impedance — Depending on the SDCFG1[OE_RULE] bit, the SD_DQS can be in high impedance until a write is occurring or only when a read is occurring.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation — Occurs on crossing of SD_CLK and $\overline{\text{SD\_CLK}}$.</td></tr>
<tr><td rowspan="3">$\overline{\text{SD\_WE}}$</td><td rowspan="3">O</td><td colspan="2">Command input. Along with $\overline{\text{SD\_CS}}$, $\overline{\text{SD\_CAS}}$, and $\overline{\text{SD\_RAS}}$ defines the current command.</td></tr>
<tr><td><b>State Meaning</b></td><td>Please see Table 21-10 for SDRAM commands.</td></tr>
<tr><td><b>Timing</b></td><td>Assertion/Negation— Occurs synchronously with SD_CLK.</td></tr>
<tr><td>SD_VREF</td><td>I</td><td colspan="2">SDRAM reference voltage. Reference voltage for differential I/O pad cells. Should be half the voltage of the memory used in the system. For example, 2.5V DDR results in an SD_VREF of 1.25V. See the device's datasheet for the voltages and tolerances for the various memory modes.</td></tr>
</table>

21.3 Interface Recommendations

21.3.1 Supported Memory Configurations

The SDRAM controller supports up to 14 row addresses and up to 13 column addresses. However, the maximum row and column addresses are not simultaneously supported. The number of row and column addresses must be less than or equal to 25. In addition to row/column address lines, there are always two row bank address bits. Therefore, the greatest possible address space accessed using a single chip select is 2^{27} x 16 bit or 256 MBytes.

Table 21-3 and Table 21-4 show the address multiplexing used by the memory controller for different configurations. When the SDRAM controller receives the internal module enable, it latches the internal bus address lines IA[27:0] (IA equals internal address) and multiplexes them into row, column, and bank addresses (RA, CA, and BA respectfully). IA[9:1] are used for CA[8:0]. IA[11:10] are used for BA[1:0], and IA[23:12] are used for RA[11:0]. IA[27:24] can be used for additional row or column address bits, as needed. The additional row- or column-address bits are programmed via the SDCR[ADDR_MUX] bits.

Table 21-2. Address Multiplexing for 16-bit Bus Mode

SDCR[ADDR_MUX]	Internal Address Bits [27:24]			
	IA[27]	IA[26]	IA[25]	IA[24]
00	CA13	CA12	CA11	CA9
01	CA12	CA11	CA9	RA12
10	CA11	CA9	RA13	RA12
11	Reserved. Do Not Use.			

Table 21-3. SDRAM-Address Multiplexing in 16-bit Bus Mode

Device	Configuration	Row bit x Col bit x Banks	SDCR [ADDR_ MUX]	Internal Address						
				27	26	25	24	23 – 12	11 – 10	9 – 1
64 Mbits	4M x 16 bit	11 x 9 x 4	00	—[1,2]	—	—	—	RA11-0	BA1-0	CA8-0
	8M x 8 bit	12 x 9 x 4	00	—	—	—	—			
	16M x 4 bit	12 x 10 x 4	00	—	—	—	CA9			
		13 x 9 x 4	01	—	—	—	RA12			
128 Mbits	8M x 16 bit	12 x 9 x 4	00	—	—	—	—	RA11-0	BA1-0	CA8-0
	16M x 8 bit	12 x 10 x 4	00	—	—	—	CA9			
		13 x 9 x 4	01	—	—	—	RA12			
	32M x 4 bit	12 x 11 x 4	00	—	—	CA11	CA9			
		13 x 10 x 4	01	—	—	CA9	RA12			
		14 x 9 x 4	10	—	—	RA13	RA12			
256 Mbits	16M x 16 bit	12 x 10 x 4	00	—	—	—	CA9	RA11-0	BA1-0	CA8-0
		13 x 9 x 4	01	—	—	—	RA12			
	32M x 8 bit	12 x 11 x 4	00	—	—	CA11	CA9			
		13 x 10 x 4	01	—	—	CA9	RA12			
		14 x 9 x 4	10	—	—	RA13	RA12			
	64M x 4 bit	12 x 12 x 4	00	—	CA12	CA11	CA9			
		13 x 11 x 4	01	—	CA11	CA9	RA12			
		14 x 10 x 4	10	—	CA9	RA13	RA12			

Table 21-3. SDRAM-Address Multiplexing in 16-bit Bus Mode (continued)

Device	Configuration	Row bit x Col bit x Banks	SDCR [ADDR_ MUX]	Internal Address						
				27	26	25	24	23 – 12	11 – 10	9 – 1
512 Mbits	32 M x 16 bit	12 x 11 x 4	00	—	—	CA11	CA9	RA11-0	BA1-0	CA8-0
		13 x 10 x 4	01	—	—	CA9	RA12			
		14 x 9 x 4	10	—	—	RA13	RA12			
	64M x 8bit	12 x 12 x 4	00	—	CA12	CA11	CA9			
		13 x 11 x 4	01	—	CA11	CA9	RA12			
		14 x 10 x 4	10	—	CA9	RA13	RA12			
1 Gbits	64M x 16bit	12 x 12 x 4	00	—	CA12	CA11	CA9	RA11-0	BA1-0	CA8-0
		13 x 11 x 4	01	—	CA11	CA9	RA12			
		14 x 10 x 4	10	—	CA9	RA13	RA12			
2 Gbits	128M x16bit	12 x 13 x 4	00	CA13	CA12	CA11	CA9	RA11-0	BA1-0	CA8-0
		13 x 12 x 4	01	CA12	CA11	CA9	RA12			
		14 x 11 x 4	10	CA11	CA9	RA13	RA12			

[1] All SD_A[13:0] bits are generated on every access, but only the bits actually used by the memory are shown.

[2] All column address (CA) bits in this table are physical column address lines. The SDRAM controller inserts an extra bit CA10 to control the precharge option.

All memory devices of a single chip-select block must have the same configuration and row/column address width; however, this is not necessary between different blocks. If mixing different memory organizations in different blocks, the following guidelines ensure that every block is fully contiguous.

- If all devices’ row address width is 12 bits, the column address can be ≥ 9 bits.
- If all devices’ row address width is 13 bits, the column address can be ≥ 9 bits.
- If all devices’ column address width is 9 bits, the row address can be ≥ 11 bits.
- The maximum row bits plus column bits equals 25.
- x16 data width memory devices cannot be mixed with any other width.

21.3.2 SDRAM DDR Component Connections

Figure 21-2 shows a block diagram using 16-bit wide DDR SDRAM (such as Micron MT46V8M16) and flash (such as Spansion AM29DBB160G).

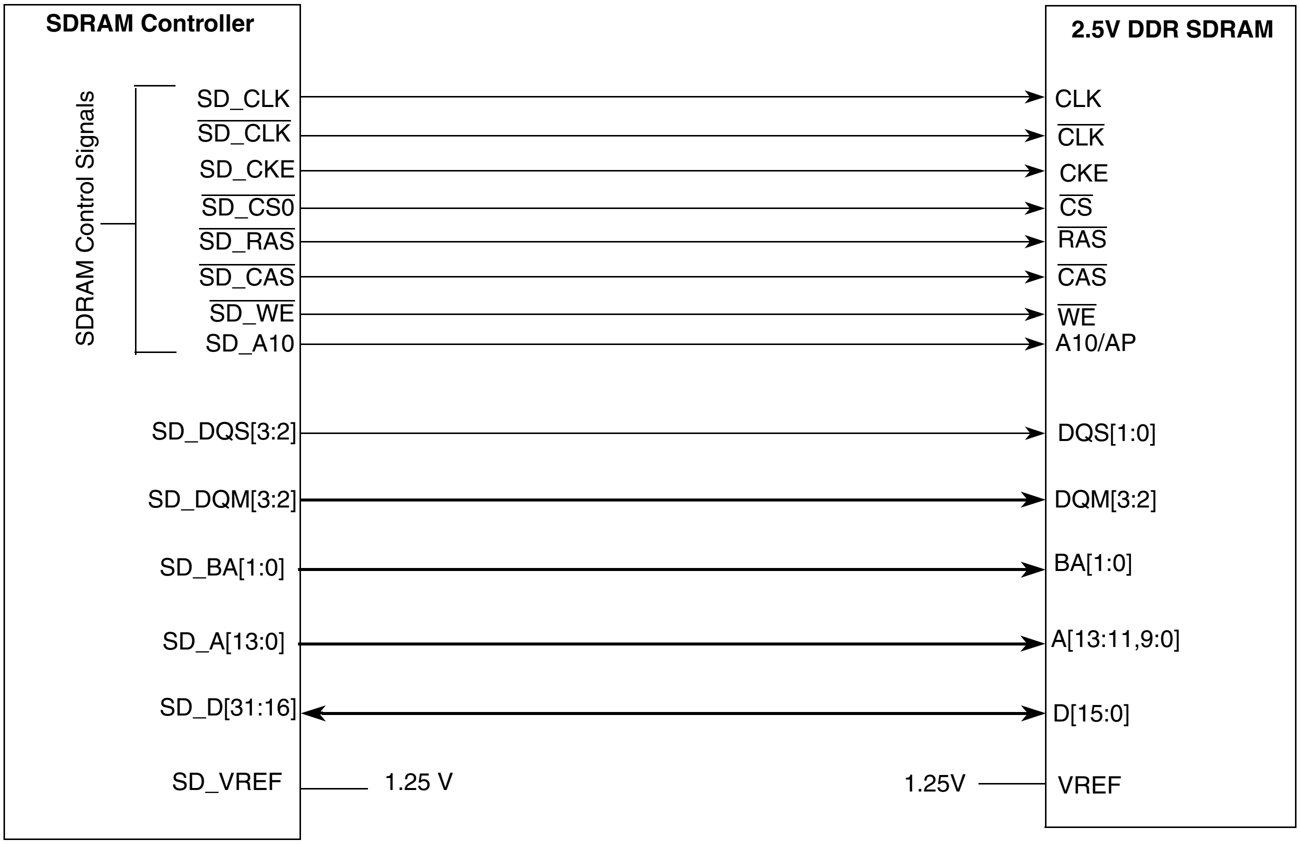


Figure 21-2. Example 2.5V, 16-bit DDR SDRAM System

21.3.3 DDR SDRAM Layout Considerations

Due to the critical timing for DDR SDRAM, a number of considerations should be taken into account during PCB layout:

- Minimize overall trace lengths.
- Each DQS, DM, and DQ group must have identical loading and similar routing to maintain timing integrity.
- Control and clock signals are routed point-to-point.
- Trace length for clock, address, and command signals should match.
- Route DDR signals on layers adjacent to the ground plane.
- Use a VREF plane under the SDRAM.
- VREF is decoupled from SDVDD and VSS.
- To avoid crosstalk, address and command signals must remain separate from data and data strobes.
- Use different resistor packs for command/address and data/data strobes.

- Use single series, single parallel termination (25 Ω series, 50 Ω parallel values are recommended, but standard resistor packs with similar values can be substituted).
- Series termination should be between the processor and memory, but closest to the processor.
- The SD_CLK and $\overline{\text{SD\_CLK}}$ signals can be terminated with a single termination resistor between the two clock phases. A 100 – 120 Ω resistor produces effective termination for the differential SD_CLK. Placement of the terminator should be physically close to the input receiver on the SDRAM(s).

  If using a SDRAM DIMM, such as a 144-pin DDR2 SO-DIMM, termination on the CLK lines is not recommended, as clock line termination is already populated on the DIMM module. Additional termination on the motherboard (main board) may cause undersired effects.
- 0.1 µF decoupling for every termination resistor pack.

21.3.3.1 Termination Example

Figure 21-3 shows the recommended termination circuitry for DDR SDRAM signals.

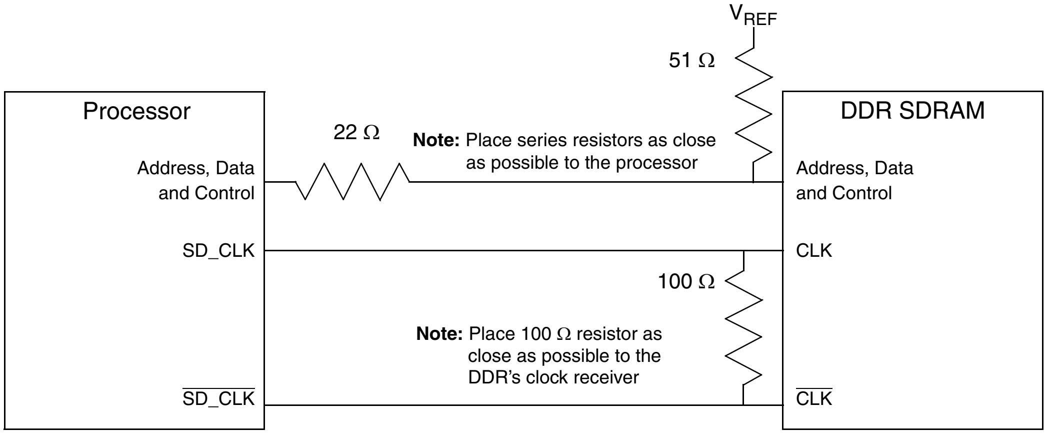

V_{REF}
51 Ω
Processor
22 Ω
Note: Place series resistors as close as possible to the processor
DDR SDRAM
Address, Data and Control
Address, Data and Control
SD_CLK
CLK
100 Ω
Note: Place 100 Ω resistor as close as possible to the DDR's clock receiver
$\overline{\text{SD\_CLK}}$
$\overline{\text{CLK}}$

Figure 21-3. DDR SDRAM Termination Circuit

21.4 Memory Map/Register Definition

The SDRAM controller and its associated logic contain two sets of programming registers:

- SDRAM controller's control and configuration registers
- Chip-select configuration control registers

NOTE

The slew rate for the SDRAM pins is controlled by a register in the pin multiplexing and control module. See Section 16.3.6, "SDRAM Mode Select Control Register (MSCR_SDRAM)," for more details.

Table 21-4 shows the SDRAM controller control and configuration registers. Unspecified memory spaces are reserved for future use. Access to reserved space is prohibited. It is recommended to write 0 to reserved space. Reads from a write-only bit return 0.

Table 21-4. SDRAMC Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_8000	SDRAM Mode/Extended Mode Register (SDMR)	32	R/W	0x0000_0000	21.4.1/21-10
0xFC0B_8004	SDRAM Control Register (SDCR)	32	R/W	0x0000_0200	21.4.2/21-11
0xFC0B_8008	SDRAM Configuration Register 1 (SDCFG1)	32	R/W	0x0000_0000	21.4.3/21-12
0xFC0B_800C	SDRAM Configuration Register 2 (SDCFG2)	32	R/W	0x0000_0000	21.4.4/21-15
0xFC0B_8110	SDRAM Chip Select 0 Configuration (SDCS0)	32	R/W	0x0000_0000	21.4.5/21-16
0xFC0B_8114	SDRAM Chip Select 1 Configuration (SDCS1)	32	R/W	0x0000_0000	21.4.5/21-16

21.4.1 SDRAM Mode/Extended Mode Register (SDMR)

The SDMR (Figure 21-4) writes to the mode and extended mode registers physically residing within the SDRAM chips. These registers must be programmed during SDRAM initialization. See Section 21.6, “Initialization/Application Information” for more information on the initialization sequence.

Address: 0xFC0B_8000 (SDMR)

Access: SDCR[MODE_EN] = 0 User read-only
SDCR[MODE_EN] = 1 User 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
R	BK		AD												0	0	0	0	DDR2_AD													
W																CMD																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-4. SDRAM-Mode/Extended-Mode Register (SDMR)

Table 21-5. SDMR Field Descriptions

Field	Description
31–30 BK	Bank address. Driven onto SD_BA[1:0] along with a LMR/LEMR command. All SDRAM chip selects are asserted simultaneously. SDCR[CKE] must be set before attempting to generate an LMR/LEMR command. The SD_BA[1:0] value is used to select between LMR and LEMR commands. 00 Load mode register command (LMR) 01 Load extended mode register command (LEMR) for non-mobile DDR devices 10 Load extended mode register command (LEMR) for mobile DDR devices 11 Reserved
29–18 AD	Address. Driven onto SD_A[11:0] along with an LMR/LEMR command. The AD value is stored as the mode (or extended mode) register data.
17	Reserved, must be cleared.
16 CMD	Command. This bit is write-only and always returns a 0 when read. 1 Generate an LMR/LEMR command 0 Do not generate any command
15–14	Reserved, must be cleared.
13–0 DDR2_AD	Address for DDR2 SDRAMs. Driven onto SD_A[13:0] along with an LMR/LEMR command. The AD value is stored as the mode (or extended mode) register data. **Note:** SDCR[DDR_MODE, DDR2_MODE] must be set for this value to appear on the bus.

21.4.2 SDRAM Control Register (SDCR)

The SDCR (Figure 21-5) controls SDRAMC operating modes, including refresh count and address line muxing.

Address: 0xFC0B_8004 (SDCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MODE_EN	CKE	DDR_MODE	REF_EN	DDR2_MODE	0	ADDR_MUX		0	OE_RULE	REF_CNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	MEM_PS	0	DQS_OE		0	0	0	0	0	0	0	0	0	DPD
W														IREF	IPALL	
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-5. SDRAM Control Register (SDCR)

Table 21-6. SDCR Field Descriptions

Field	Description
31 MODE_EN	SDRAM mode register programming enable. 0 SDMR locked, cannot be written. 1 SDMR enabled, can be written. **Note:** MODE_EN must be cleared during normal operation,
30 CKE	Clock enable. CKE must be set to perform normal read and write operations. Clear CKE to put the memory in self-refresh or power-down mode. 0 SD_CKE is negated (low) 1 SD_CKE is asserted (high)
29 DDR_MODE	DDR mode select. Reserved 1 DDR mode
28 REF_EN	Refresh enable. 0 Automatic refresh disabled 1 Automatic refresh enabled
27 DDR2_MODE	DDR2 mode select. 0 DDR mode 1 DDR2 mode **Note:** If DDR_MODE is cleared, this bit is ignored.
26	Reserved, must be cleared.
25–24 ADDR_MUX	Controls the use of internal address bits A[27:24] as row or column bits on the SD_A bus. See Table 21-2, and Table 21-3.
23	Reserved, must be cleared.
22 OE_RULE	Drive rule selection. 0 Tri-state except to write. SD_D and SD_DQS are only driven when necessary to perform a write command. 1 Drive except to read. SD_D and SD_DQS are only tristated when necessary to perform a read command. When not being driven for a write cycle, SD_D hold the most recent value and SD_DQS are driven low. This mode is intended for minimal applications only, to prevent floating signals and allow unterminated board traces. However, terminated wiring is always recommended over unterminated.

Table 21-6. SDCR Field Descriptions (continued)

Field	Description
21–16 REF_CNT	The average periodic interval at which the controller generates refresh commands to memory; measured in increments of 64 × SD_CLK period. REF_CNT = (t_{REFI}/ (t_{CK} × 64)) - 1, rounded down to the next integer value. If the SDRAM data sheet does not define t_{REFI}, it can be calculated by t_{REFI} = t_{REF} / #rows.
15–14	Reserved, must be cleared.
13 MEM_PS	Memory data port size. 0 Reserved 1 16-bit data bus
12	Reserved, must be cleared.
11–10 DQS_OE	DQS output enable. Each bit of the DQS_OE field is a master enable for the corresponding SD_DQS*n* signal. DQS_OE[1] (SDCR[11]) enables SD_DQS3 and DQS_OE[0] (SDCR[10]) enables SD_DQS2. 0 SD_DQS*n* can never drive. Use this value in DDR mode with a single DQS memory. Some 32-bit DDR devices have only a single DQS pin. Enable one of the SD_DQS*n* signals and disable the other. Then, short both pins external to the device. 1 SD_DQS*n* can drive as necessary, depending on commands and SDCR[OE_RULE] setting. DDR only.
9–3	Reserved, must be cleared.
2 IREF	Initiate refresh command. Used to force a software-initiated refresh command. This bit is write-only, reads return zero. 0 Do not generate a refresh command. 1 Generate a refresh command. All $\overline{\text{SD\_CS}n}$ signals are asserted simultaneously. SDCR[CKE] must be set before attempting to generate a software refresh command. **Note:** A software requested refresh is completely independent of the periodic refresh interval counter. Software refresh is only possible when MODE_EN is set.
1 IPALL	Initiate precharge all command. Used to force a software-initiated precharge all command. This bit is write-only, reads return zero. 0 Do not generate a precharge command. 1 Generate a precharge all command. All $\overline{\text{SD\_CS}n}$ signals are asserted simultaneously. SDCR[CKE] must be set before generating a software precharge command. **Note:** Software precharge is only possible when MODE_EN is set. **Note:** Do not set IREF and IPALL at the same time.
0 DPD	Deep power-down mode. This bit is only applicable for mobile-DDR (LPDDR) devices. 0 No effect or exit from deep power down-mode 1 Generate a deep power-down mode command. All $\overline{\text{SD\_CS}n}$ signals are asserted simultaneously

21.4.3 SDRAM Configuration Register 1 (SDCFG1)

The 32-bit read/write SDRAM configuration register 1 (SDCFG1) stores necessary delay values between specific SDRAM commands. During initialization, software loads values to the register according to the selected SD_CLK frequency and SDRAM information obtained from the data sheet. This register resets only by a power-up reset signal.

The read and write latency fields govern the relative timing of commands and data and must be exact values. All other fields govern the relative timing from one command to another; they have minimum values, but any larger value is also legal (but with decreased performance).

For DDR2, burst reads and writes are the normal operation. The DDR2 specification does not use the BURST TERMINATE command; therefore, the entire burst occurs. If the burst length is eight and a 32-bit write is occurring, the first two cycles have data, while the last six cycles have the data mask disabled.

The minimum values of certain fields can be different for DDR, and DDR2 SDRAM, even if the data sheet timing is the same, because:

- In DDR mode, the memory controller counts the delay in 2 x SD_CLK (also referred to as SD_CLK2)
- In DDR2 mode, the memory controller counts the delay in SD_CLK (this is not SD_CLK2).
- SD_CLK—memory controller clock—is the speed of the SDRAM interface and is equal to the internal bus clock.
- SD_CLK2—double frequency of SD_CLK—DDR uses both edges of the bus-frequency clock (SD_CLK) to read/write data

NOTE

In all calculations for setting the fields of this register, convert time units to clock units and round up to the nearest integer.

Address: 0xFC0B_8008 (SDCFG1) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	SRD2RWP				0	SWT2RWP			RD_LAT				0	ACT2RW		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	PRE2ACT			REF2ACT				0	WT_LAT			0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-6. SDRAM Configuration Register 1 (SDCFG1)

Table 21-7. SDCFG1 Field Descriptions

Field	Description
31–28 SRD2RWP	Single read to read/write/precharge delay. Limiting case is read to write. DDR: SRD2RWP = CL + 1 DDR2: SRD2RWP = BurstLength/2 + AL; If AL (additive latency) is less than 2, then use 2. t_{HZ} is the time the data bus uses to return to hi-impedance after a read and is found in the SDRAM device specifications. **Note:** Count value is in SD_CLK periods for DDR mode.
27	Reserved, must be cleared.

Table 21-7. SDCFG1 Field Descriptions (continued)

Field	Description
26–24 SWT2RWP	Single write to read/write/precharge delay. Limiting case is write to precharge. DDR: SWT2RWP = t_{WR} + 1 DDR2: SWT2RWP = CL + AL + t_{WR} - 1 **Note:** Count value is in SD_CLK periods for DDR mode.
23–20 RD_LAT	Read CAS Latency. Read command to read data available delay counter. For DDR2, RD_LAT is replaced with the CAS latency. For DDR: If CL = 2, write 0x6 If CL = 2.5, write 0x7 The recommended values are just a starting point and may need to be adjusted depending on the trace length for the data and DQS lines.DDR: Count value is in SD_CLK2 periods. **Note:** DDR2: Count value is in SD_CLK periods.
19	Reserved, must be cleared.
18–16 ACT2RW	Active to read/write delay. Active command to any following read- or write-delay counter. Suggested value = ($t_{RCD} \times f_{SD\_CLK}$) - 1 (Round up to nearest integer) Example: If t_{RCD} = 20ns and f_{SD_CLK} = 99 MHz Suggested value = (20ns × 99 MHz) - 1= 0.98; round to 1. **Note:** Count value is in SD_CLK periods for DDR1/2 modes.
15	Reserved, must be cleared.
14–12 PRE2ACT	Precharge to active delay. Precharge command to following active command delay counter. Suggested value = ($t_{RP} \times f_{SD\_CLK}$) - 1 (Round up to nearest integer) Example: If t_{RP} = 20ns and f_{SD_CLK} = 99 MHz Suggested value = (20ns × 99 MHz) - 1 = 0.98; round to 1. **Note:** Count value is in SD_CLK periods for DDR1/2 modes.
11–8 REF2ACT	Refresh to active delay. Refresh command to following active or refresh command delay counter. DDR: REF2ACT = ($t_{RFC} \times f_{SD\_CLK}$) - 1 (Round up to nearest integer) DDR2: REF2ACT = $t_{RFC}/(t_{CLK} \times 2)$ (Round up to the nearest integer) Example (for DDR): If t_{RFC} = 75ns and f_{SD_CLK} = 99 MHz Suggested value = (75ns × 99 MHz) - 1 = 6.425; round to 7. **Note:** Count value is in SD_CLK periods for DDR1/2 modes.
7	Reserved, must be cleared.

Table 21-7. SDCFG1 Field Descriptions (continued)

Field	Description
6–4 WT_LAT	Write latency. Write command to write data delay counter. DDR: write 0x3 DDR2: write the additive latency (AL) value DDR mode: Count value is in SD_CLK2 periods.
3–0	Reserved, must be cleared.

21.4.4 SDRAM Configuration Register 2 (SDCFG2)

The 32-bit read/write configuration register 2 stores delay values necessary between specific SDRAM commands. During initialization, software loads values to the register according to the SDRAM information obtained from the data sheet. This register is reset only by a power-up reset signal.

The burst length (BL) field must be exact. All other fields govern the relative timing from one command to another, they have minimum values but any larger value is also legal (but with decreased performance).

All delays in this register are expressed in SD_CLK. In all calculations for setting the fields of this register, convert time units to clock units and round up to the nearest integer.

Address: 0xFC0B_800C (SCFG2) Access: User 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
R	BRD2RP				BWT2RWP				BRD2W				BL				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-7. SDRAM Configuration Register 2 (SDCFG2)

Table 21-8. SDCFG2 Field Descriptions

Field	Description
31–28 BRD2RP	Burst read to read/precharge delay. Limiting case is read to read. DDR: BRD2RP = BurstLength/2 + 1 DDR2: BRD2RP = BurstLength/2 + AL (Limiting case for DDR2 is Read to Precharge.)
27–24 BWT2RWP	Burst write to read/write/precharge delay. Limiting case is write to precharge. DDR: BWT2RWP = BurstLength/2 + t_{WR} DDR2: BWT2RWP = CL + AL + BurstLength/2 + t_{WR} - 1
23–20 BRD2W	Burst read to write delay. DDR: BRD2W = CL + BurstLength/2 - 1 DDR2: BRD2W = BurstLength/2 + 2. Suggested value = 0x6.
19–16 BL	Burst length. BL = BurstLength - 1 **Note:** Burst length depends on port size. If 16-bit bus (SDCR[MEM_PS] = 1), burst length is 8. Write BL = 7.
15–0	Reserved, must be cleared.

21.4.5 SDRAM Chip Select Configuration Registers (SDCS*n*)

These registers define base address and space size of each chip select.

NOTE

> Because the SDRAM module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x4000_0000 – 0x7FFF_FFFF. Be sure to set the SDCS*n* registers appropriately.

NOTE

> The user should not probe memory on a DDR chip select to determine if memory is connected. If a read is attempted from a DDR SDRAM chip select when there is no memory to respond with the appropriate DQS pulses, the bus cycle hangs. Because no high level bus monitor exists on the device, a reset is the only way to exit the error condition.

Address: 0xFC0B_8110 (SDCS0)
0xFC0B_8114 (SDCS1)

Access: User 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
R	CSBA												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CSSZ				
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-8. SRAM Chip Select Configuration Register (SDCS*n*)

Table 21-9. SDCS*n* Field Descriptions

Field	Description
31–20 CSBA	Chip-select base address. Because the SDRAM module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x4000_0000 – 0x7FFF_FFFF. Therefore, the possible range for this field is 0x400 – 0x7FF.

Table 21-9. SDCS*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>19–5</td><td>Reserved, must be cleared.</td></tr>
<tr><td>4–0
CSSZ</td><td>Chip select size.
<table>
<tr><th>CSSZ</th><th>Size</th><th>Address Lines to Compare</th><th>CSSZ</th><th>Size</th><th>Address Lines to Compare</th></tr>
<tr><td>00000</td><td>Disabled</td><td>—</td><td>11000</td><td>32 MByte</td><td>A[31:25]</td></tr>
<tr><td>00001–10001</td><td>Reserved</td><td>Reserved</td><td>11001</td><td>64 MByte</td><td>A[31:26]</td></tr>
<tr><td>10010</td><td>Reserved</td><td>Reserved</td><td>11010</td><td>128 MByte</td><td>A[31:27]</td></tr>
<tr><td>10011</td><td>1 MByte</td><td>A[31:20]</td><td>11011</td><td>256 MByte</td><td>A[31:28]</td></tr>
<tr><td>10100</td><td>2 MByte</td><td>A[31:21]</td><td>11100</td><td>512 MByte</td><td>A[31:29]</td></tr>
<tr><td>10101</td><td>4 MByte</td><td>A[31:22]</td><td>11101</td><td>1 GByte</td><td>A[31:30]</td></tr>
<tr><td>10110</td><td>8 MByte</td><td>A[31:23]</td><td>11110</td><td>2 GByte</td><td>A31</td></tr>
<tr><td>10111</td><td>16 MByte</td><td>A[31:24]</td><td>11111</td><td>4 GByte</td><td>Ignore A[31:20]</td></tr>
</table>
</td></tr>
</table>

Any chip-select can be enabled or disabled, independent of others. Any chip-select can be allocated any size of address space from 1M to 4G, independent of others. Any chip-select address space can begin at any size-aligned base address, independent of others.

For contiguous memory with different sizes of memory blocks, place largest block at the lowest address and place smaller blocks in descending size order at ascending base addresses.

For example, assume a system with 2 chip selects: CS0=16M, CS1=256M:

CS0CFG = 0x4F000017 = enable 16M @ 0x4F000000-0x4FFFFFFF

CS1CFG = 0x5000001B = enable 256M @ 0x50000000-0x5FFFFFFF

This gives 272 MB total memory, at 0x4F000000-0x5FFFFFFF.

21.5 Functional Description

21.5.1 SDRAM Commands

When an internal bus master accesses SDRAM address space, the memory controller generates the corresponding SDRAM command. Table 21-10 lists SDRAM commands supported by the memory controller.

Table 21-10. SDRAM Commands

Function	Symbol	CKE	$\overline{\text{CS}}$	$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	BA[1:0]	A[10]	Other A
Command Inhibit	INH	H	H	X	X	X	X	X	X
No Operation	NOP	H	L	H	H	H	X	X	X

Table 21-10. SDRAM Commands (continued)

Function	Symbol	CKE	$\overline{\text{CS}}$	$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	BA[1:0]	A[10]	Other A
Row and Bank Active	ACTV	H	L	L	H	H	V	V	V
Read	READ	H	L	H	L	H	V	L	V
Write	WRITE	H	L	H	L	L	V	L	V
Burst Terminate (DDR only)	BST	H	L	H	H	L	X	X	X
Precharge All Banks	PALL	H	L	L	H	L	X	H	X
Precharge Selected Bank	PRE	H	L	L	H	L	V	L	X
Load Mode Register	LMR	H	L	L	L	L	LL	V	V
Load Extended Mode Register	LEMR	H	L	L	L	L	LH	V	V
Auto Refresh	REF	H	L	L	L	H	X	X	X
Self Refresh	SREF	H→L	L	L	L	H	X	X	X
Power Down	PDWN	H→L	H	X	X	X	X	X	X
Deep Power-Down	DPD	L	L	H	H	L	X	X	X
H = High L = Low V = Valid X = Don’t care									

Many commands require a delay before the next command may be issued; sometimes the delay depends on the type of the next command. These delay requirements are managed by the values programmed in the memory controller configuration registers (SDCFG1, SDCFG2).

21.5.1.1 Row and Bank Active Command (ACTV)

The ACTV command is responsible for latching the row and bank address and activating the specified row bank of a memory block. After the row is activated, it can be accessed using subsequent read and write commands.

NOTE

The SDRAMC supports one active row for each chip select block. See Section 21.6.4, “Page Management,” for more information.

21.5.1.2 Read Command (READ)

When the SDRAMC receives a read request via the internal bus, it first checks the row and bank of the new access. If the address falls within the active row of an active bank, it is a page hit, and the read is issued as soon as possible (pending any delays required by previous commands). If the address is within an inactive bank, the memory controller issues an ACTV followed by the read command. If the address is not within the active row of an active bank, the memory controller issues a pre command to close the active row. Then, the SDRAMC issues ACTV to activate the necessary row and bank for the new access, followed by the read to the SDRAM.

The PALL/PRE and ACTV commands (if necessary) can sometimes be issued in parallel with an on-going data movement.

To truncate a burst read when only a single read is needed, the memory controller issues the burst-terminate command. With DDR memory, the data masks are asserted high throughout the entire read size; but DDR memory ignores the data masks during reads. With DDR2 memory, burst terminate commands are not recognized and are not given by the SDRAM controller.

21.5.1.3 Write Command (WRITE)

When the memory controller receives a write request via the internal bus, it first checks the row and bank of the new access. If the address falls within the active row of an active bank, it is a page hit, and the WRITE is issued as soon as possible (pending any delays required by previous commands). If the address is within the inactive bank, the memory controller issues an ACTV followed by the write command. If the address is not within the active row of an active bank, the memory controller issues a PRE command to close the active row. Then, the SDRAMC issues ACTV to activate the necessary row and bank for the new access, followed by the WRITE command to the SDRAM.

The PALL/PRE and ACTV commands (if necessary) can sometimes be issued in parallel with an on-going data movement.

With DDR memory, a read command can be issued overlapping the masked beats at the end of a previous single write of the case $\overline{\text{CS}}$; the read command aborts the remaining (unnecessary) write beats.

21.5.1.4 Burst-Terminate Command (BST)

SDRAMs are burst-only devices, but provide mechanisms to truncate a burst if all of the beats are not needed. The burst-terminate command truncates read bursts. To truncate a burst write for DDR, the read command can abort the remaining unnecessary write beats. The most recently registered read or write command prior to the burst terminate command is truncated. The active page remains open.

21.5.1.5 Precharge-All-Banks (PALL) and Selected-Bank (PRE) Commands

The precharge command puts SDRAM into an idle state. The SDRAM must be in this idle state before a REF, LMR, LEMR, or ACTV command to open a new row within a particular bank can be issued.

The memory controller issues the precharge command only when necessary for one of these conditions:

- Access to a new row
- Refresh interval elapsed
- Software commanded precharge during device initialization

NOTE

A precharge is required after DRAMs also have a maximum bank-open period. The memory controller does not time the bank-open period because the refresh interval is always less.

21.5.1.6 Load Mode/Extended Mode Register Command (LMR, LEMR)

All SDRAM devices contain mode registers that configure the timing and burst mode for the SDRAM. These commands access the mode registers that physically reside within the SDRAM devices. During the LMR or LEMR command, SDRAM latches the address and bank buses to load the values into the selected mode register.

NOTE

The LMR and LEMR commands are only used during SDRAM initialization.

Use the following steps to write the mode register and extended mode register:

1. Set the SDCR[MODE_EN] bit.
2. Write the SDMR[BA] bits to select the mode register.
3. Write the desired mode register value to the SDMR[ADDR]. Do not overwrite the SDMR[BA] values. This step can be performed in the same register write in step 2.
4. Set the SDMR[CMD] bit.
5. For DDR, perform steps 2–4 more than once to write the extended-mode register and the mode register.
   For DDR2, perform step 2–4 four times. The first is for the third extended mode register, second is for the second extended mode register; third is for the first extended mode register; the last is for the mode register.
6. Clear the SDCR[MODE_EN] bit.

21.5.1.6.1 Mode Register Definition

Figure 21-9 shows a typical mode register definition. This is the SDRAM's mode register, not the SDRAMC's mode/extended mode register (SDMR) defined in Section 21.4.1, "SDRAM Mode/Extended Mode Register (SDMR)." Refer to the SDRAM manufacturer's device data sheet to confirm correct settings.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	0	OP_MODE					CL			BT	BLEN		

Figure 21-9. Typical Mode Register

Table 21-11. Mode Register Field Descriptions

Field	Description
BA1–BA0	Bank address. These must be zero to select the mode register.
A11–A7 OP_MODE	Operating mode. 00000 Normal Operation 00010 Reset DLL Else Reserved
A6–A4 CL	CAS latency. Delay in clocks from issuing a READ to valid data out. Check the SDRAM manufacturer's spec because the CL settings supported can vary from memory to memory.

Table 21-11. Mode Register Field Descriptions (continued)

Field	Description
A3 BT	Burst type. 0 Sequential 1 Interleaved. This setting should not be used because the SDRAMC does not support interleaved bursts.
A2–A0 BLEN	Burst length. Determines the number of column locations that are accessed for a given READ or WRITE command. 001 Two 010 Four 011 Eight Else Reserved

21.5.1.6.2 DDR Extended Mode Register Definition

Figure 21-11 shows a typical extended-mode register used by DDR SDRAMs. This is the SDRAM’s extended mode register, not the SDRAMC’s mode/extended-mode register (SDMR) defined in Section 21.4.1, “SDRAM Mode/Extended Mode Register (SDMR).” Refer to the SDRAM manufacturer’s device data sheet to confirm correct settings.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	1	OPTION											DLL

Figure 21-10. Typical DDR Extended Mode Register

Table 21-12. Typical DDR Extended-Mode Register Field Descriptions

Field	Description
BA1–BA0	Bank address. These must be set to 01 to select the extended mode register.
A11–A1 OPTION	Option. These bits are not defined by the DDR specification. Each DDR SDRAM manufacturer can use these bits to implement optional features. Check with the SDRAM manufacturer to determine if any optional features have been implemented. For normal operation all bits must be cleared.
A0 DLL	Delay locked loop. Controls enabling of the delay locked loop circuitry used for DDR timing. 0 Enabled 1 Disabled

21.5.1.6.3 Low-Power/Mobile DDR Extended Mode Register Definition

Figure 21-11 shows a typical extended-mode register used by low-power/mobile DDR SDRAMs. This is the SDRAM’s extended mode register, not the SDRAMC’s mode/extended-mode register (SDMR) defined in Section 21.4.1, “SDRAM Mode/Extended Mode Register (SDMR).” Refer to the SDRAM manufacturer’s device data sheet to confirm correct settings.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	1	0	—					DS		TCSR		PASR		

Figure 21-11. Typical Mobile DDR Extended Mode Register

Table 21-13. Mobile DDR Extended-Mode Register Field Descriptions

Field	Description
BA1–BA0	Bank address. These must be set to 10 to select the extended mode register.
A11–A7	Reserved, must be cleared.
A6–A5 DS	Drive strength. 00 Full strength 01 Half strength 10 Quarter strength 11 One-eighth strength
A4–A3 TCSR	Temperature-compensated self-refresh. Check the SDRAM manufacturer's spec because the use of the TCSR settings can vary from memory to memory.
A2–A0 PASR	Partial array self refresh coverage. 000 Full array 001 Half array 010 Quarter array 101 One-eighth array 110 One-sixteenth array All other settings are reserved.

21.5.1.6.4 DDR2 Mode Register Definition

Figure 21-12 shows the mode register used by DDR2 SDRAMs. This is the SDRAM's extended mode register, and not the SDRAMC's mode/extended mode register (SDMR) described in Section 21.4.1, "SDRAM Mode/Extended Mode Register (SDMR)".

	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	0	PD	WR			DLL	TM	CL			BT	BL		

Figure 21-12. DDR2 Mode Register

Table 21-14. DDR2 Mode Register Field Descriptions

Field	Description
BA1–BA0	Bank address. These must be zero to select the mode register
A12 PD	Active Power Down exit time. 0 Use Fast exit (Use t_{XARD}) 1 Use Slow exit (Use t_{XARDS})
A11–A9 WR	Write recovery for precharge. The t_{CK} max determines WR_{min} and t_{CK} min determines WR_{max}. WR in clock cycles is calculated by dividing t_{WR} (in ns) by t_{CK} (in ns) and rounding up to the next integer (WR[cycles] = t_{WR}(ns)/t_{CK}(ns)). The mode register must be programmed to this value. This is also used with t_{RP} to determine t_{DAL}. 000 Reserved 001 2 010 3 011 4 100 5 101 6 11x Reserved

Table 21-14. DDR2 Mode Register Field Descriptions (continued)

Field	Description
A8 DLL	DLL mode 0 Normal 1 Reset DLL
A7 TM	Test mode 0 Normal 1 Test Mode
A6–A4 CL	CAS latency. Delay in clocks from issuing a READ to valid data. Check the SDRAM manufacturer's spec as the CL settings supported can vary from memory to memory.
A3 BT	Burst type 0 Sequential 1 Interleaved. This setting should not be used because the SDRAMC does not support interleaved bursts.
A2–A0 BL	Burst length 010 4 011 8

21.5.1.6.5 DDR2 Extended Mode Register Set (1)

Figure 21-13 shows the extended mode register used by DDR2 SDRAMs. This is the SDRAM's extended mode register, and not the SDRAMC's mode/extended mode register (SDMR) described in Section 21.4.1, "SDRAM Mode/Extended Mode Register (SDMR)".

	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	1	Q$_{\text{off}}$	RDQS	$\overline{\text{DQS}}$	OCD program			R$_{\text{tt}}$	Additive Latency			R$_{\text{tt}}$	D.I.C.	DLL

Figure 21-13. DDR2 Extended Mode Register (1)

Table 21-15. DDR2 Extended Mode Register (1) Field Descriptions

Field	Description
BA1–BA0	Bank address. Must be set to 01 to select the DDR2 Extended Mode Register (1).
A12 Q$_{\text{off}}$	Output buffer disabled. Disables DQs, DQSs, $\overline{\text{DQS}}$s, RDQS, $\overline{\text{RDQS}}$. This feature is used with dimm IDD measurements when IDDQ is not desired to be included. 0 Output buffer enabled 1 Output buffer disabled
A11 RDQS	RDQS enable. If RDQS is enabled, DM function is disabled. RDQS is active for reads and don't care for writes. Used in conjunction with DQS to produce the strobe function matrix in Table 21-16. 0 Disable 1 Enable.
A10 DQS	$\overline{\text{DQS}}$ enable. Enables the $\overline{\text{DQS}}$ signals to be used for data strobes. 0 Enable 1 Disable

Table 21-15. DDR2 Extended Mode Register (1) Field Descriptions (continued)

Field	Description
A9–A7 OCD program	OCD calibration program. 000 OCD calibration mode exit; maintain setting 001 Drive (1) 010 Drive (0) 100 Adjust mode 111 OCD calibration default.
A6,A2 R_{tt}	On die termination 00 ODT Disabled 01 75 ohm 10 150 ohm 11 Reserved
A5–A3	Additive latency. 000 0 001 1 010 2 011 3 100 4 Else Reserved
A1	Output driver impedance control 0 Normal, Driver size of 100% 1 Weak, Driver size of 60%
A0	DLL enable 0 DLL Enable 1 DLL Disable

Table 21-16. Strobe-Function Matrix

A11 (RDQS Enable)	A10 ($\overline{\text{DQS}}$ Enable)	Strobe Function Matrix			
		RDQS/DM	RDQS	DQS	$\overline{\text{DQS}}$
0 (Disable)	0 (Enable)	DM	Hi-Z	DQS	$\overline{\text{DQS}}$
0 (Disable)	1 (Disable)	DM	Hi-Z	DQS	Hi-Z
1 (Enable)	0 (Enable)	RDQS	RDQS	DQS	$\overline{\text{DQS}}$
1 (Enable)	1 (Disable)	RDQS	Hi-Z	DQS	Hi-Z

21.5.1.6.6 DDR2 Extended Mode Register (2) and (3)

Figure 21-14 shows the extended mode register 2 and 3 used by DDR2 SDRAMs. This is the SDRAM's extended mode register, and not the SDRAMC's mode/extended mode register (SDMR) described in Section 21.4.1, "SDRAM Mode/Extended Mode Register (SDMR)".

	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	1	0	0												

Figure 21-14. DDR2 Extended Mode Register 2

	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	1	1	0												

Figure 21-15. DDR2 Extended Mode Register 3

Table 21-17. DDR2 Extended Mode Register 2 and 3 Field Descriptions

Field	Description
BA1–BA0	Bank Address. BA1 must be set to 1. To select DDR2 extended mode register 2, BA0 has to be a 0. To select DDR2 extended mode register 3, BA0 has to be a 1.
A12–A0	Reserved for future use. All bits except for BA1 and BA0 must be programmed to 0 when setting the mode register during initialization.

21.5.1.7 Auto-Refresh Command (REF)

The memory controller issues auto-refresh commands according to the SDCR[REF_CNT] value. Each time the programmed refresh interval elapses, the memory controller issues a PALL command followed by a REF command.

If a memory access is in progress at the time the refresh interval elapses, the memory controller schedules the refresh after the transfer finishes; the interval timer continues counting so the average refresh rate is constant.

After REF command, the SDRAM is in an idle state and waits for an ACTV command.

21.5.1.8 Self-Refresh (SREF) and Power Down (PDWN) Commands

The memory controller issues a PDWN or a SREF command if the SDCR[CKE] bit is cleared. If the SDCR[REF_EN] bit is set when CKE is negated, the controller issues a SREF command; if the REF_EN bit is cleared, the controller issues a PDWN command. The REF_EN bit may be changed in the same register write that changes the CKE bit; the controller acts upon the new value of the REF_EN bit.

Like an auto-refresh command, the controller automatically issues a PALL command before the self-refresh command.

The memory reactivates from power-down or self-refresh mode by setting the CKE bit.

If a normal refresh interval elapses while the memory is in self-refresh mode, a PALL and REF performs when the memory reactivates. If the memory is put into and brought out of self-refresh all within a single-refresh interval, the next automatic refresh occurs on schedule.

In self-refresh mode, memory does not require an external clock. The SD_CLK can be stopped for maximum power savings. If the memory controller clock is stopped, the refresh-interval timer must be reset before the memory is reactivated (if periodic refresh is to be resumed). The refresh-interval timer resets by clearing the REF_EN bit. This can be done at any time while the memory is in self-refresh mode, before or after the memory controller clock is stopped/restarted, but *not* with the same control register write that clears CKE; this would put the memory in power down mode. To restart periodic refresh when the memory reactivates, the REF_EN bit must be reasserted; this can be done before the memory reactivates or in the same control register write that sets CKE to exit self-refresh mode.

21.5.1.9 Deep Power-Down (DPD) Mode

Deep power-down mode is available for mobile-DDR (LPDDR) devices to turn the SDRAM into its lowest power state possible. Deep power-down mode eliminates power to the entire memory array. Array data is not retained after the device enters deep power-down mode.

Deep power-down mode is entered by having all banks idle, then asserting $\overline{\text{CS}}$ and $\overline{\text{WE}}$ with $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ negated at the rising edge of the clock, while CKE is negated. CKE must be negated during deep power-down.

To exit deep power-down mode, CKE must be asserted. After exiting, the following sequence is needed to enter a new command: maintain NOP input conditions for a minimum of 100 μs, issue precharge commands for all banks, issue eight or more auto-refresh commands. The values of the mode register and the extended mode register are retained upon exiting deep power-down.

21.5.2 Read Clock Recovery (RCR) Block

The RCR block allows the external DDR memory devices to generate clock pulses (strobes) that define the data valid window for each DDR data cycle. The RCR delay block compensates for each byte lane and generates an internal read strobe targeted to the center of the data valid window provided by the external DDR memories.

Figure 21-16 displays a simple timing diagram that illustrates the end result of the RCR delay.

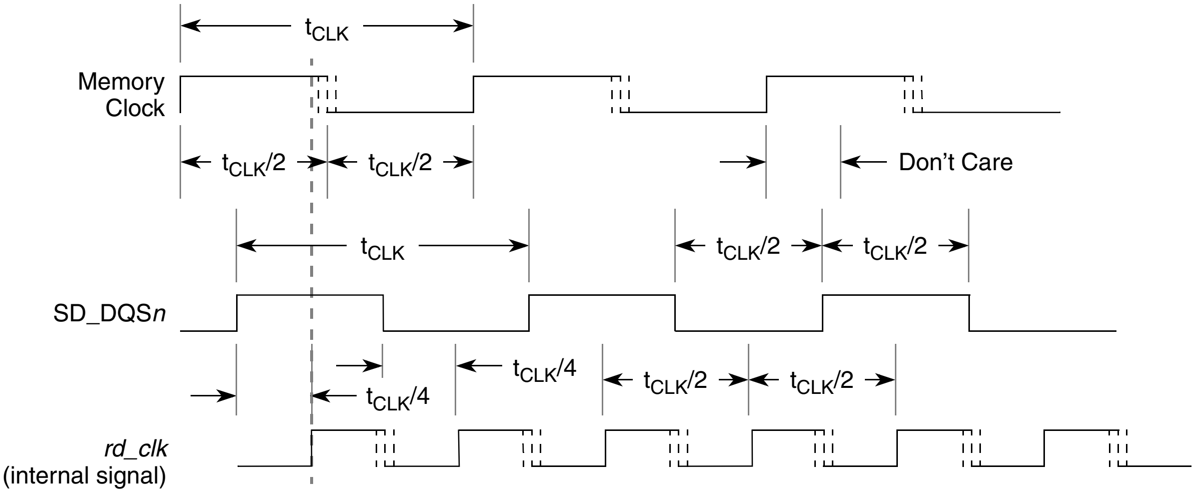


Figure 21-16. Frequency Doubler Block Diagram

Dual data rate (DDR) memories provide data strobe (DQS) timing reference signals in parallel with read data. However, these strobe signals cannot directly clock the data because the strobe edges are aligned with the edges of the data valid window, not the center. The RCR delay module is responsible for delaying the received DQS edges to achieve data-center alignment instead of data-edge alignment. There are two data valid windows per memory clock period with DDR, so the nominal delay of read clocks from DQS is 1/4 memory clock period.

The RCR delay module maintains the 1/4 memory clock period delay of the DQS signals across the full range of silicon process, voltage, and temperature conditions.

The RD_CLK is an internal reconstructed clock derived from DQS. It is twice the frequency of DQS, with the rising edge shifted 1/4 memory clock period after the DQS edge to align with the nominal center of the data valid window.

21.6 Initialization/Application Information

SDRAMs have a prescribed initialization sequence. The following sections detail the memory initialization steps for DDR SDRAM, mobile DDR, and DDR2 SDRAM. The sequence might change ghtly from device-to-device. Refer to the SDRAM manufacturer's device datasheet as the most relevant reference.

21.6.1 DDR SDRAM Initialization Sequence

1. After reset is deactivated, pause for the amount of time indicated in SDRAM specification. Usually 200µs.
2. Configure pin multiplex control for shared $\overline{\text{SD\_CS}}$ pins in pin multiplexing and control module if needed.
3. Configure the slew rate for the SDRAM external pins in the pin multiplexing and control module's MSCR_SDRAM register if needed.
4. Write the SDCS*n* register values for each chip select that is used.
5. Program SDRAM configuration registers (SDCFG1 and SDCFG2) with correct delay and timing values.
6. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
7. Initialize the SDRAM's extended mode register to enable the DLL. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for instructions on issuing a LEMR command.
8. Initialize the SDRAM's mode register and reset the DLL using the LMR command. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing a LMR command. During this step the OP_MODE field of the mode register should be set to normal operation/reset DLL.
9. Pause for the DLL lock time specified by the memory.
10. Issue a second PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
11. Refresh the SDRAM. The SDRAM specification should indicate a number of refresh cycles to be performed before issuing an LMR command (usually two). Write to the SDCR with the IREF and MODE_EN bits set (SDCR[REF and IPALL] must be cleared). This forces a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.

12. Initialize the SDRAM's mode register using the LMR command. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LMR command. During this step the OP_MODE field of the mode register should be set to normal operation.
13. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

21.6.2 Low-power/Mobile SDRAM Initialization Sequence

1. After reset is deactivated, pause for the amount of time indicated in SDRAM specification. Usually 100µs or 200µs.
2. Configure pin multiplex control for shared $\overline{\text{SD\_CS}}$ pins in pin multiplexing and control module.
3. Configure the slew rate for the SDRAM external pins in the pin multiplexing and control module's MSCR_SDRAM register.
4. Write the base address and mask registers (SDBAR0, SDBAR1, SDMR0, and SDMR1) to setup the address space for each chip-select.
5. Program SDRAM configuration registers (SDCFG1 and SDCFG2) with correct delay and timing values.
6. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
7. Refresh the SDRAM. The SDRAM specification should indicate a number of refresh cycles to be performed before issuing an LMR command (usually two). Write to the SDCR with the IREF and MODE_EN bits set (SDCR[REF and IPALL] must be cleared). This forces a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.
8. Initialize the SDRAM's mode register using the LMR command. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LMR command.
9. Initialize the SDRAM's extended mode register using the LEMR command. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LEMR command.
10. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

21.6.3 DDR2 SDRAM Initialization Sequence

1. After reset is deactivated, pause for the amount of time indicated in SDRAM specification. Usually 200µs.
2. Configure pin multiplex control for shared $\overline{\text{SD\_CS}}$ pins in pin multiplexing and control module if needed.
3. Configure the slew rate for the SDRAM external pins in the pin multiplexing and control module's MSCR_SDRAM register if needed.
4. Write the SDCS*n* register values for each chip select that is used.

5. Program SDRAM configuration registers (SDCFG1 and SDCFG2) with correct delay and timing values.
6. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
7. For DDR2, write zeroes to the extended mode registers 2 and 3.
8. Initialize the SDRAM's extended mode register to enable the DLL. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for instructions on issuing a LEMR command.
9. Initialize the SDRAM's mode register and reset the DLL using the LMR command. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing a LMR command. During this step the OP_MODE field of the mode register should be set to normal operation/reset DLL.
10. Pause for the DLL lock time specified by the memory.
11. Issue a second PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
12. Refresh the SDRAM. The SDRAM specification should indicate a number of refresh cycles to be performed before issuing an LMR command (usually two). Write to the SDCR with the IREF and MODE_EN bits set (SDCR[REF and IPALL] must be cleared). This forces a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.
13. Initialize the SDRAM's mode register using the LMR command. See Section 21.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LMR command. During this step the OP_MODE field of the mode register should be set to normal operation.
14. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

21.6.4 Page Management

SDRAM devices have four internal banks. A particular row and bank of memory must be activated to allow read and write accesses. The SDRAM controller supports paging mode to maximize the memory access throughout. During operation, the SDRAM controller maintains an open page for each $\overline{\text{SD\_CS}}$ block. An open page is composed of the active rows in the internal banks. Each internal bank has its own active row.

The physical page size of a $\overline{\text{SD\_CS}}$ block is equal to the space size divided by the number of rows; but the page may not be contiguous in the internal address space because SDRAMs can have a different row address open in each bank and the internal address bits (A[27:24] and A[9:2]) or (A[27:24] and A[9:1]) used for memory column addresses are not consecutive.

Because the column address may split across two portions of the internal address, the contiguous page size is (number of contiguous columns per bank) × (number of bits). This gives a contiguous page size of 1 KBytes. However, the total (possibly fragmented) page size is (number of banks) × (number of columns) × (number of bits).

If a new access does not fall in the open row of an open bank of a $\overline{\text{SD\_CS}}$ block, the open row must be closed (PRE) and the new row must be opened (ACTV), then the READ or WRITE command can proceed. An ACTV command activates only one bank at one time. If another read or write falls in an inactive bank, another ACTV is needed, but no precharge is needed. If a read or write falls in any open row of any active banks of a page, no PRE or ACTV is needed; the read or write command can be issued immediately.

A page is kept open until one of the following conditions occurs:

- An access outside the open page.
- A refresh cycle is started.

All $\overline{\text{SD\_CS}}$ blocks are refreshed at the same time. The refresh closes all banks of every SDRAM block.

21.6.5 Transfer Size

In the SDRAMC, the internal data bus is 32 bits wide, while the SDRAM external interface bus is 16 bits wide. Therefore, each internal data beat requires two memory data beats. The SDRAM controller manages the size translation (packing/unpacking) between internal and external DRAM buses.

The burst size is the processor standard 16 bytes: Four beats of 4 bytes on the internal bus or eight beats of 2 bytes (16-bit mode) on the memory bus. The SDRAM controller follows the critical beat first, sequential transfer format required.

The burst size and transfer order must be programmed in the SDRAM mode registers during initialization; the burst size also must be programmed in the memory controller (SDCFG2 register).

Chapter 22
PCI Bus Controller

22.1 Introduction

This chapter details the operation of the PCI bus controller for the device. The PCI bus arbiter, included in this module, is also detailed here. The PCI controller acts as a bridge between the PCI bus and the internal system of the device. Figure 22-1 shows a high-level diagram that illustrates the functional organization of the block.

NOTE

The MCF54450 and MCF54451 devices do not contain a PCI bus controller.

22.1.1 Block Diagram

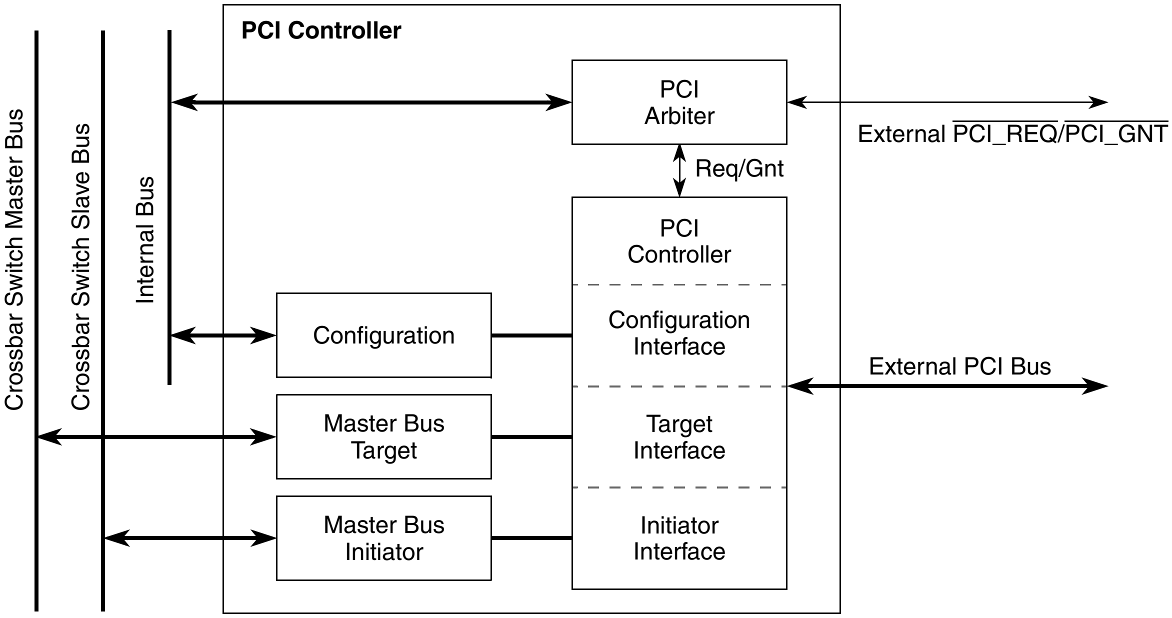


Figure 22-1. PCI Block Diagram

22.1.2 Overview

The peripheral component interface (PCI) bus is a high-performance bus with multiplexed address and data lines, especially suitable for high data-rate applications.

The PCI controller module supports a 32-bit PCI initiator (master) and target interface. As a target, access to the internal bus is supported. As an initiator, the PCI controller is coupled directly to the crossbar switch (as a slave).

The device contains PCI central resource functions such as the PCI arbiter and PCI reset control. The PCI bus clock must be provided by an external source. It must be phase aligned and either equal to 1, 1/2, 1/3, 2/3, 1/4, or 1/5 the frequency of the system clock.

22.1.3 Features

The following PCI features are supported:

- Supports system clock: PCI clock frequency ratios 1:1, 2:1, 3:1, 3:2, 4:1, and 5:1
- Uses external CLKIN as clock reference
- Compatible with PCI 2.2 specification
- PCI initiator and target operation
- Fully synchronous design
- 32-bit PCI address bus
- PCI 2.2 Type 0 configuration space header
- Supports the PCI 16/8 clock rule
- Host and agent mode support
- On-chip arbitration with support for up to 4 request and grant pairs
- PCI to system bus address translation
- Target response is medium $\overline{\text{PCI\_DEVSEL}}$ generation
- Initiator latency time-outs
- Memory prefetching of inbound PCI read accesses
- Supports posting of outbound and inbound PCI memory writes
- Automatic retry of target disconnects

22.1.4 Modes of Operation

A host/agent configuration parameter is provided at reset as a reset configuration signal. See the Chapter 11, “Chip Configuration Module (CCM),” for more information on reset configuration. This parameter selects between host and agent mode for the PCI interface. When operating as a PCI host bridge, no PCI interrupt is provided and all six BAR registers are enabled. In agent mode, one interrupt output is provided out of reset and BAR registers default to disabled. Target retries are automatically issued for any incoming PCI configuration accesses to allow for initialization software to set up the BAR configuration before PCI enumeration. The host/agent mode selection is determined at power-up.

The internal PCI arbiter can be statically configured as enabled or disabled. When disabled, an external arbiter is responsible for PCI arbitration and the PCI controller’s request and grant signals are presented as external pins of the device.

22.2 External Signal Description

A list of the PCI external signals and their properties are discussed below. For detailed description of the PCI bus signals, see the *PCI Local Bus Specification, Revision 2.2*.

Table 22-1. PCI Module External Signals

Name	Secondary[1]	Function	Type	Reset
PCI_AD[31:0]	—	PCI Address Data Bus	I/O	Tristate
PCI_CLK	—	PCI Clock	I	Toggling
PCI_CBE[3:0]	—	PCI Command/Bytes Enables	I/O	Tristate
PCI_DEVSEL	—	PCI Device Select	I/O	Tristate
PCI_FRAME	—	PCI Frame	I/O	Tristate
PCI_GNT[3:0]	PCI_REQOUT	PCI Grants/Request Out	O	Tristate
PCI_IDSEL	—	PCI Initialization Device Select	I	Tristate
PCI_INTA	—	PCI Interrupt A	O	0
PCI_IRDY	—	PCI Initiator Ready	I/O	Tristate
PCI_PAR	—	PCI Parity	I/O	Tristate
PCI_PERR	—	PCI Parity Error	I/O	Tristate
PCI_REQ[3:0]	PCI_GNTIN	PCI Requests/Grant In	I	Tristate
PCI_RST	—	PCI Reset	O	0
PCI_SERR	—	PCI System Error	I/O	Tristate
PCI_STOP	—	PCI Stop	I/O	Tristate
PCI_TRDY	—	PCI Target Ready	I/O	Tristate

[1] Arbitration signal function when PCI Arbiter is disabled.

22.2.1 Address/Data Bus (PCI_AD[31:0])

The PCI_AD[31:0] lines are a time multiplexed address/data bus. The address is presented on the bus during the address phase while the data is presented on the bus during one or more data phases.

22.2.2 Clock (PCI_CLK)

The PCI_CLK signal serves as a reference clock for generation of the internal PCI clock. For more information, see Section 22.4.6, “PCI Clock Scheme.”

22.2.3 Command/Byte Enables (PCI_CBE[3:0])

The PCI_CBE[3:0] signals are time multiplexed. The PCI command is presented during the address phase and the byte enables are presented during the data phase. Byte enables are active low.

22.2.4 Device Select (PCI_DEVSEL)

The PCI_DEVSEL signal asserts active low when the PCI controller decodes it is the target of a PCI transaction from the address presented on the PCI bus during the address phase.

22.2.5 Frame (PCI_FRAME)

The PCI_FRAME signal asserts active low by a PCI initiator to indicate the beginning of a transaction. It deasserts when the initiator can complete the final data phase.

22.2.6 Grant (PCI_GNT[3:0])

The PCI_GNT[3:0] signals assert to external masters to give them control of the PCI bus. When the PCI arbiter module is disabled, PCI_GNT[0] operates as the request output of the PCI controller and PCI_GNT[3:1] is statically driven high.

22.2.7 Initialization Device Select (PCI_IDSEL)

The PCI_IDSEL signal asserts active high during a PCI Type 0 configuration cycle to address the PCI configuration header.

22.2.8 Interrupt (PCI_INTA)

The PCI_INTA signal asserts active low by the PCI controller to request an interrupt.

22.2.9 Initiator Ready (PCI_IRDY)

The PCI_IRDY signal is asserted active low to indicate that the PCI initiator can transfer data. During a write operation, assertion indicates the master is driving valid data on the bus. During a read operation, assertion indicates the master can accept data.

22.2.10 Parity (PCI_PAR)

The PCI_PAR signal indicates the parity on the PCI_AD[31:0] and PCI_CBE[3:0] lines.

22.2.11 Parity Error (PCI_PERR)

If enabled, the PCI_PERR signal asserts active low when a data phase parity error is detected.

22.2.12 Request (PCI_REQ[3:0])

The PCI_REQ[3:0] signals assert by external PCI masters when they require access to the PCI bus. When the internal PCI arbiter module is disabled, PCI_REQ[0] is the grant input for the PCI controller. It is driven by an external PCI arbiter.

22.2.13 Reset ($\overline{\text{PCI\_RST}}$)

The $\overline{\text{PCI\_RST}}$ signal asserts active low by the PCI controller to reset the PCI bus. This signal asserts after device reset and must be negated to enable usage of the PCI bus.

22.2.14 System Error ($\overline{\text{PCI\_SERR}}$)

The $\overline{\text{PCI\_SERR}}$ signal, if enabled, asserts active low when an address phase parity error is detected.

22.2.15 Stop ($\overline{\text{PCI\_STOP}}$)

The $\overline{\text{PCI\_STOP}}$ signal asserts active low by the currently addressed target to indicate it wishes to stop the current transaction.

22.2.16 Target Ready ($\overline{\text{PCI\_TRDY}}$)

The $\overline{\text{PCI\_TRDY}}$ signal asserts active low by the currently addressed target to indicate it can complete the current data phase.

22.3 Memory Map/Register Definition

The device has several sets of registers that control and report status for the different interfaces to the PCI controller: PCI Type 0 configuration space registers, general status/control registers, and PCI arbiter registers. As an internal bus master, an external PCI bus master can access internal memory space for register updates.

PCIGSCR[PR] bit controls $\overline{\text{PCI\_RST}}$, and must first be cleared before external PCI devices wake-up. In other words, an external PCI master cannot load configuration software across the PCI bus until software clears this bit. Access to all internal registers is supported regardless of the value held in PCIGSCR[PR].

Software reads from unimplemented registers return 0x0000_0000 and writes have no effect.

NOTE

The slew rate for the PCI pins is controlled by a register in the GPIO module. Please see Section 16.3.7, "PCI Mode Select Control Register (MSCR_PCI)", for more details.

Table 22-2. PCI Controller Memory Map

Address	Register	Width	Access	Reset Value	Section/Page
PCI Type 0 Configuration Registers					
0xFC0A_8000	PCI Device ID/Vendor ID (PCIIDR)	32	R	0x5807_1957	22.3.1.1/22-7
0xFC0A_8004	PCI Status/Command (PCISCR)	32	R/W	0x02A0_0000	22.3.1.2/22-8
0xFC0A_8008	PCI Class Code/Revision ID (PCICCRIR)	32	R	0x0680_0000	22.3.1.3/22-10
0xFC0A_800C	PCI Configuration 1 Register (PCICR1)	32	R/W	0x0000_0000	22.3.1.4/22-10
0xFC0A_8010	PCI Base Address Register 0 (PCIBAR0)	32	R/W	0x0000_0000	22.3.1.5/22-11

Table 22-2. PCI Controller Memory Map (continued)

Address	Register	Width	Access	Reset Value	Section/Page
0xFC0A_8014	PCI Base Address Register 1 (PCIBAR1)	32	R/W	0x0000_0008	22.3.1.5/22-11
0xFC0A_8018	PCI Base Address Register 2 (PCIBAR2)	32	R/W	0x0000_0008	22.3.1.5/22-11
0xFC0A_801C	PCI Base Address Register 3 (PCIBAR3)	32	R/W	0x0000_0008	22.3.1.5/22-11
0xFC0A_8020	PCI Base Address Register 4 (PCIBAR4)	32	R/W	0x0000_0008	22.3.1.5/22-11
0xFC0A_8024	PCI Base Address Register 5 (PCIBAR5)	32	R/W	0x0000_0008	22.3.1.5/22-11
0xFC0A_8028	PCI Cardbus CIS Pointer (PCICCPR)	32	R/W	0x0000_0000	22.3.1.6/22-12
0xFC0A_802C	PCI Subsystem ID/Subsystem Vendor ID (PCISID)	32	R/W	0x0000_0000	22.3.1.7/22-13
0xFC0A_8030	PCI Expansion ROM Base Address Registers (PCIERBAR)	32	R/W	0x0000_0000	22.3.1.8/22-13
0xFC0A_8034	PCI Capabilities Pointer Register (PCICPR)	32	R/W	0x0000_0000	22.3.1.9/22-13
0xFC0A_803C	PCI Configuration Register 2 (PCICR2)	32	R/W	0x0000_0100	22.3.1.10/22-14
	General Control/Status Registers				
0xFC0A_8060	PCI Global Status/Control Register (PCIGSCR)	32	R/W	See Section	22.3.2.1/22-15
0xFC0A_8064	PCI Target Base Address Translation Register 0 (PCITBATR0)[1]	32	R/W	0x0000_0000	22.3.2.2/22-16
0xFC0A_8068	PCI Target Base Address Translation Register 1 (PCITBATR1)[2]	32	R/W	0x0000_0000	22.3.2.3/22-17
0xFC0A_806C	PCI Target Control 1 Register (PCITCR1)	32	R/W	0x0000_0008	22.3.2.4/22-18
0xFC0A_8070	PCI Initiator Window 0 Base/Translation Address Register (PCIIW0BTAR)	32	R/W	0x0000_0000	22.3.2.5/22-18
0xFC0A_8074	PCI Initiator Window 1 Base/Translation Address Register (PCIIW1BTAR)	32	R/W	0x0000_0000	22.3.2.5/22-18
0xFC0A_8078	PCI Initiator Window 2 Base/Translation Address Register (PCIIW2BTAR)	32	R/W	0x0000_0000	22.3.2.5/22-18
0xFC0A_8080	PCI Initiator Window Configuration Register (PCIIWCR)	32	R/W	0x0000_0000	22.3.2.6/22-19
0xFC0A_8084	PCI Initiator Control Register (PCIICR)	32	R/W	0x0000_00FF	22.3.2.7/22-20
0xFC0A_8088	PCI Initiator Status Register (PCIISR)	32	R/W	0x0000_0000	22.3.2.8/22-21
0xFC0A_808C	PCI Target Control 2 Register (PCITCR2)	32	R/W	See Section	22.3.2.9/22-22
0xFC0A_8090	PCI Target Base Address Translation Register 0 (PCITBATR0)	32	R/W	0x0000_0000	22.3.2.10/22-23
0xFC0A_8094	PCI Target Base Address Translation Register 1 (PCITBATR1)	32	R/W	0x0000_0000	22.3.2.10/22-23
0xFC0A_8098	PCI Target Base Address Translation Register 2 (PCITBATR2)	32	R/W	0x0000_0000	22.3.2.10/22-23
0xFC0A_809C	PCI Target Base Address Translation Register 3 (PCITBATR3)	32	R/W	0x0000_0000	22.3.2.10/22-23
0xFC0A_80A0	PCI Target Base Address Translation Register 4 (PCITBATR4)	32	R/W	0x0000_0000	22.3.2.10/22-23

Table 22-2. PCI Controller Memory Map (continued)

Address	Register	Width	Access	Reset Value	Section/Page
0xFC0A_80A4	PCI Target Base Address Translation Register 5 (PCITBATR5)	32	R/W	0x0000_0000	22.3.2.10/22-23
0xFC0A_80A8	PCI Interrupt Register (PCIINTR)	32	R/W	0x0000_0000	22.3.2.11/22-24
0xFC0A_80F8	PCI Configuration Address Register (PCICAR)	32	R/W	0x0000_0000	22.3.2.12/22-25
PCI Arbiter Registers					
0xFC0A_C000	PCI Arbiter Control Register (PACR)	32	R/W	0x8000_0000	22.3.3.1/22-26
0xFC0A_C004	PCI Arbiter Status Register (PASR)	32	R/W	0x0000_0000	22.3.3.2/22-27

[1] Alias of PCTBATR0 at location 0xFC0A_8090.

[2] Alias of PCTBATR1 at location 0xFC0A_8094.

22.3.1 PCI Type 0 Configuration Registers

The PCI controller supplies a type 0 PCI configuration space header. These registers are accessible as an offset within the device memory map or through externally mastered PCI configuration cycles. Only external PCI configuration accesses can access PCI Dword reserved space (0x10 – 0x3F).

22.3.1.1 Device ID/Vendor ID Register (PCIIDR)—PCI Dword Addr 0

Address: 0xFC0A_8000 (PCIIDR) Access: User 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
R	Device ID																Vendor ID															
W																																
Reset	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	1	0	1	0	1	0	1	1	1

Figure 22-2. PCIIDR Register

Table 22-3. PCIIDR Field Descriptions

Field	Description
31–16 Device ID	This field is read-only and represents the PCI Device ID assigned to this processor. Its value is: 0x5807.
15–0 Vendor ID	This field is read-only and represents the PCI Vendor ID assigned to this processor. Its value is: 0x1957.

22.3.1.2 PCI Status/Command Register (PCISCR)—PCI Dword Addr 1

Address: 0xFC0A_8004 (PCISCR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PE	SE	MA	TR	TS	DT		DP	FC	R	66M	C	0	0	0	0
W	w1c[1]	w1c[1]	w1c[1]	w1c[1]	w1c[1]			w1c[1]								
Reset	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	F	S	ST	PER	V	MW	SP	B	M	IO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Bits 31-27 and 24 are write-one-to-clear (w1c).
Hardware can set w1c bits, but cannot clear them.
Only PCI configuration cycles can clear w1c bits that are currently set by writing a 1 to the bit location. Writing a 1 to a w1c bit that is currently a 0 or writing a 0 to any w1c bit has no effect.

Figure 22-3. PCISCR Register

Table 22-4. PCISCR Field Descriptions

Field	Description
31 PE	Parity error detected. This bit is set when a parity error is detected, even if the PCISCR[PER] bit is cleared. The PCI controller checks parity for address or data during the following cycles: 1. Address phase driven by an external master. 2. Write data transferred to the PCI Controller from an external master. 3. Read data transferred to the PCI Controller when operating as a PCI master. If a data parity error is detected, this bit is set; if the PCISCR[PER] bit is set, the $\overline{\text{PCI\_PERR}}$ signal is asserted. A PCI configuration cycle writing a 1 to the bit clears it. Writing 0 has no effect.
30 SE	System error signalled. This bit is set when the PCI controller generates a PCI system error on the $\overline{\text{PCI\_SERR}}$ signal. A PCI configuration cycle writing a 1 to the bit clears it. Writing 0 has no effect.
29 MA	Master abort received. Set when the PCI controller is the PCI master and terminates a transaction (except for a special cycle) with a master-abort. A PCI configuration cycle writing a 1 to the bit clears it. Writing 0 has no effect.
28 TR	Target abort received. Set when the PCI controller is the PCI master and a transaction terminates by a target-abort from the currently addressed target. A PCI configuration cycle writing a 1 to the bit clears it.Writing 0 has no effect.
27 TS	Target abort signalled. Set when the PCI controller is the target and it terminates a transaction with a target-abort. A PCI configuration cycle writing a 1 to the bit clears it. Writing 0 has no effect.
26–25 DT	$\overline{\text{PCI\_DEVSEL}}$ timing. Fixed to 01. These bits encode a medium $\overline{\text{PCI\_DEVSEL}}$ timing. This defines the slowest $\overline{\text{PCI\_DEVSEL}}$ timing as medium timing when the PCI controller is the target (except configuration accesses).
24 DP	Master data parity error. Applies only when the PCI controller is the master and is set only if: • The PCI controller-as-master asserts $\overline{\text{PCI\_PERR}}$ itself during a read or the PCI controller-as-master detected it asserted by the target during a write • The PCISCR[PER] bit is set A PCI configuration cycle writing a 1 to the bit clears it. Writing 0 has no effect.
23 FC	Fast back-to-back capable. Fixed to 1. This read-only bit indicates that the PCI controller as target is capable of accepting fast back-to-back transactions with other targets.

Table 22-4. PCISCR Field Descriptions (continued)

Field	Description
22 R	Reserved. Fixed to 0. Prior to the 2.2 PCI specification, this was the UDF (user defined features) supported bit. 0 Does not support UDF 1 Supported user defined features
21 66M	66 MHz capable. Fixed to 1. Indicates the PCI controller is 66 MHz capable.
20 C	Capabilities list. Fixed to 0. Indicates the PCI controller does not implement the new capabilities list pointer configuration register in DWORD 13 of the configuration space.
19–10	Reserved, must be cleared.
9 F	Fast back-to-back transfer enable. Controls if the PCI controller as master can do fast back-to-back transactions to different devices. Initialization software should set this bit if all targets are fast back-to-back capable. 0 Fast back-to-back transactions only allowed to the same device 1 The master is allowed to generate fast back-to-back transactions to different devices
8 S	SERR enable. Enables the PCI_SERR driver. 0 PCI_SERR driver disabled 1 PCI_SERR driver enabled **Note:** Address parity errors are reported only if this bit and the PER bit are set.
7 ST	Address and data stepping. Fixed to 0. Indicates the PCI controller never uses address/data stepping. Initialization software should write a 0 to this bit location.
6 PER	Parity error response. Controls the device's response to parity errors. 0 The device sets its parity error status bit (PE, bit 31) in the event of a parity error, but does not assert PCI_PERR 1 When a parity error is detected, the PCI controller asserts PCI_PERR
5 V	VGA palette snoop enable. Fixed to 0. Indicates that the PCI controller is not VGA compatible. Initialization software should write a 0 to this bit location.
4 MW	Memory write and invalidate enable. Enables the MEMORY WRITE AND INVALIDATE command. 0 Only MEMORY WRITE command can be used 1 PCI controller-as-master may generate the MEMORY WRITE AND INVALIDATE command
3 SP	Special cycle monitor or ignore. Determines whether or not to ignore PCI special cycles. Because PCI controller-as-target does not recognize messages delivered via the special cycle operation, a value of 1 must never be programmed to this register. This bit, however, is programmable.
2 B	Bus master enable. Indicates if the PCI controller has the ability to serve as a master on the PCI bus. A value of 1 indicates this ability is enabled. If the PCI controller is a master on the PCI bus, a 1 must be written to this bit during initialization or the PCI controller does not operate as a PCI master. Configuration software reads this bit.
1 M	Memory access control. Controls the PCI controller's response to memory space accesses. 0 The PCI controller does not recognize memory accesses 1 The PCI controller recognizes memory accesses
0 IO	I/O access control. Fixed to 0. This bit is not implemented because there is no PCI controller I/O type space accessible from the PCI bus. The PCI base address registers are memory address ranges only. Initialization software must write a 0 to this bit location.

22.3.1.3 Revision ID/Class Code Register (PCICCRIR)—PCI Dword 2

Address: 0xFC0A_8008 (PCICCRIR) Access: User 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
R	Class Code																								Revision ID							
W																																
Reset	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-4. PCICCRIR Register

Table 22-5. PCICCRIR Field Descriptions

Field	Description
31–8 Class Code	This field is read-only and represents the PCI Class Code assigned to processor. Its value is: 0x06_8000. (Other bridge device).
7–0 Revision ID	This field is read-only and represents the PCI Revision ID for this version of the processor. Its value is: 0x00.

22.3.1.4 Configuration 1 Register (PCICR1)—PCI Dword 3

Address: 0xFC0A_800C (PCICR1) Access: User 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
R	BIST								Header Type								LTMR[7:3]					LTMR[2:0]			CLS[7:4]				CLS[3:0]			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-5. PCICR1 Register

Table 22-6. PCICR1 Field Descriptions

Field	Description
31–24 BIST	Built in self test. Fixed to 0x00. The PCI controller does not implement the built-in self-test register. Initialization software should write a 0x00 to this register location.
23–16 Header Type	Header type. Fixed to 0x00. The PCI controller implements a Type 0 PCI configuration space header. Initialization software must write a 0x00 to this register location.
15–8 LTMR	Latency timer [7:3]. Contains the latency timer value, in PCI clocks, used when the PCI controller is the PCI master. The upper five bits are programmable. Latency timer must be programmed to a non-zero value before the PCI controller operates as master of the PCI bus.
	Latency timer [2:0] The lower three bits of the bit field are hardwired low.
7–0 CLS	Cache line size[7:4] Specifies the cache line size in units of DWORDs. The higher four bits of the bit field are hardwired low
	Cache line size[3:0] Specifies the cache line size in units of DWORDs.

22.3.1.5 Base Address Register *n* (PCIBAR*n*)—PCI Dword 4–9

Accessibility of the following six registers, PCIBAR0–5, is controlled by six enable register bits, PCITCR2[B*n*E]. When a PCIBAR*n* register is not enabled, all 32-bits of the registers are read-only. The low bits remain as specified and the high bits, the base address bits, cannot be written to. When disabled, the value of the PCIBAR*n* registers are not used to decode the PCI address during PCI memory cycles.

Address: 0xFC0A_8010 (PCIBAR0) Access: User 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
R	BAR0														0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREF	RANGE		IO/M#
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-6. PCIBAR0 Register

Address: 0xFC0A_8014 (PCIBAR1) Access: User 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
R	BAR1												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREF	RANGE		IO/M#
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-7. PCIBAR1 Register

Address: 0xFC0A_8018 (PCIBAR2) Access: User 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
R	BAR2										0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREF	RANGE		IO/M#
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-8. PCIBAR2 Register

Address: 0xFC0A_801C (PCIBAR3) Access: User 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
R	BAR3								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREF	RANGE		IO/M#
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-9. PCIBAR3 Register

Address: 0xFC0A_8020 (PCIBAR4) Access: User 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
R	BAR4					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREF	RANGE		IO/M#
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-10. PCIBAR4 Register

Address: 0xFC0A_8024 (PCIBAR5)

Access: User 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
R	BAR5			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PREF	RANGE		IO/M#
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-11. PCIBAR5 Register

Table 22-7. PCIBAR*n* Field Descriptions

Field	Description
See Figures[1] BAR0	Base address register *n*. Applies only when the PCI controller is target. These bits are programmable. | BAT*n* | Bit Field Width | Boundary Size | | BAT0 | 14 | 256 Kbye | | BAT1 | 12 | 1 Mbyte | | BAT2 | 10 | 4 Mbyte | | BAT3 | 8 | 16 Mbyte | | BAT4 | 5 | 128 Mbyte | | BAT5 | 3 | 512 Mbyte |
See Figures[1]	Reserved, must be cleared.
3 PREF	Prefetchable access. Indicates if memory space defined by BAR0 is prefetchable. For PCIBAR0 this value is fixed to 0, while the other PCIBAR registers are fixed to 1. 0 Un-prefetchable 1 Prefetchable
2–1 RANGE	Fixed to 00. Indicates the base address is 32 bits wide and can be mapped anywhere in 32-bit address space. Configuration software should write 00 to these bit locations.
0 IO/M#	IO or memory space. Fixed to 0. Indicates that the base address is for memory space. Configuration software should write a 0 to this bit location. 0 Memory 1 I/O

[1] See corresponding PCIBAR*n* figure above for bit numbers for the BAR*n* and reserved bit fields.

22.3.1.6 CardBus CIS Pointer Register (PCICCPR)—PCI Dword A

This optional register contains the pointer to the card information structure (CIS) for the CardBus card. All 32 bits of the register are programmable by the slave bus. This register can only be read from the PCI bus, not written.

Address: 0xFC0A_8028 (PCICCPR) Access: User 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
R W	Pointer																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-12. PCICCPR Register

Table 22-8. PCICCPR Field Descriptions

Field	Description
31–0 Pointer	Pointer to the card information structure (CIS) for the CardBus card.

22.3.1.7 Subsystem ID/Subsystem Vendor ID Registers (PCISID)—PCI Dword B

The subsystem vendor ID register contains the manufacturer and subsystem identification number of the add-in board or subsystem that contains this PCI device. A 0 value in these registers indicates no subsystem vendor and subsystem ID associated with the device. If used, software must write to these registers before any PCI bus master reads them. All 32 bits of the register are programmable by the slave bus. From the PCI bus, this register can only be read, not written.

Address: 0xFC0A_802C (PCISID) Access: User 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
R W	Subsystem ID																Subsystem Vendor ID															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-13. PCISID Register

Table 22-9. PCISID Field Descriptions

Field	Description
31–16 Subsystem ID	Indicates the subsystem identification number of the add-in board or subsystem containing this PCI device.
15–0 Subsystem Vendor ID	Indicates the manufacturer identification number of the add-in board or subsystem containing this PCI device.

22.3.1.8 Expansion ROM Base Address (PCIERBAR)—PCI Dword C

Not implemented. Fixed to 0x0000_0000 at address 0xFC0A_8030.

22.3.1.9 Capabilities Pointer (Cap_Ptr) (PCICPR)—PCI Dword D

Not implemented. Fixed to 0x0000_0000 at address 0xFC0A_8034.

22.3.1.10 Configuration 2 Register (PCICR2)—PCI Dword F

Address: 0xFC0A_803C (PCICR2)

Access: User 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
R	Max_Lat								Min_Gnt								InterruptPin								InterruptLine							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

Figure 22-14. PCICR2 Register

Table 22-10. PCICR2 Field Descriptions

Field	Description
31–24 Max_Lat	Maximum latency. Specifies how often, in units of 1/4 microseconds, the PCI controller would like to have access to the PCI bus as master. A 0 value indicates the device has no stringent requirement in this area. This field is programmable from the internal bus, but read only from the PCI bus.
23–16 Min_Gnt	Minimum grant. Indicates how long the PCI controller as master would like to retain PCI bus ownership when it initiates a transaction. The register is programmable from the internal bus, but read only from the PCI bus.
15–8 Interrupt Pin	Interrupt pin. Indicates the number of interrupt pins the PCI controller uses. Fixed to 0x01 which indicates $\overline{\text{PCI\_INTA}}$ is used.
7–0 Interrupt Line	Interrupt line. Stores a value that identifies which input on a PCI interrupt controller to which the PCI interrupt request pin is routed.

22.3.2 General Control/Status Registers

The general control/status registers primarily address the configurability of the internal bus initiator and target interfaces, though some also address global options which affect the DMA interface. These registers are accessed primarily internally, but can also be accessed by an external PCI master if PCI base and target base address registers are configured to access the space. See Section 22.5.2, "Address Translation," on configuring address windows.

22.3.2.1 Global Status/Control Register (PCIGSCR)

Address: 0xFC0A_8060 (PCIGSCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DRD	0	PE	SE	ER	AUTODIV			0	0	0	0	0	PRGDIV		
W	w1c[1]		w1c[1]	w1c[1]	w1c[1]											
Reset	0	0	0	0	0	—[2]	—[2]	—[2]	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DRDE	0	PEE	SEE	ERE	0	0	0	0	0	0	0	0	0	0	PR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

[1] Bits 31,29 and 28 are write-one-to-clear (w1c).
Hardware can set w1c bits, but cannot clear them.
Software can clear w1c bits currently set by writing a 1 to the bit location. Writing a 1 to a w1c bit currently a 0 or writing a 0 to any w1c bit has no effect.

[2] The reset value of bits 26-24 and 18-16 is determined by the PLL multiplier.

Figure 22-15. PCIGSCR Register

Table 22-11. PCIGSCR Field Descriptions

Field	Description
31 DRD	Delayed read discarded. Set when an internally completed delayed read is discarded due to a non-compliant or tardy external PCI master. The delayed buffer clears because the master that initiated a target read transaction to the PCI_BAR*n* register has not retried the transaction in 2^{15} (32,768) PCI clocks. Clearing the buffer allows other target reads to procede to PCI_BAR*n* target space. Setting of this bit can trigger an interrupt request to the processor if the PCIGSCR[DRDE] bit is set.
30	Reserved, must be cleared.
29 PE	$\overline{\text{PCI\_PERR}}$ detected. Set when the PCI parity error signal, $\overline{\text{PCI\_PERR}}$, asserts (any device). A CPU interrupt is generated if the PCIGSCR[PEE] bit is set. It is up to application software to clear this bit by writing 1 to it.
28 SE	SERR detected. Set when a PCI system error signal, $\overline{\text{PCI\_SERR}}$, asserts (any device). An interrupt is generated if the PCIGSCR[SEE] bit is set. It is up to application software to clear this bit by writing 1 to it.
27 ER	Error response detected. Set when an internal error occurs during a PCI target transaction. An interrupt is generated if the PCIGSCR[ERE] bit is set. It is up to application software to clear this bit by writing 1 to it.
26–24 AUTODIV	Auto-detected clock divide. Stores automatically detected internal bus clock to external PCI clock divide ratio. This field is read-only and the reset value is determined by the ratio detected. Software must read this bit to determine whether or not the auto-detect logic is functioning correctly. If the register contains a differential value that does not reflect the PLL settings, PCI controller could malfunction. 000 Reserved 001 Divide by 1 010 Divide by 2 011 Divide by 3 100 Divide by 4 101 Divide by 5 110 Divide by 1.5 (Multiply by $^2/_3$) 111 Reserved
23–19	Reserved, must be cleared.

Table 22-11. PCIGSCR Field Descriptions (continued)

Field	Description
18–16 PRGDIV	Programmable clock divide. Stores programmable internal bus clock to external PCI clock divide ratio. When this bit field is non-zero, this ratio has priority over the auto-detected clock ratio indicated in the AUTODIV field. When set to a reserved setting, PCI controller could malfunction. Software can revert back to the auto-detected divide ratio by clearing this bit field. 000 Ignore bit field and use AUTODIV 001 Divide by 1 010 Divide by 2 011 Divide by 3 100 Divide by 4 101 Divide by 5 110 Divide by 1.5 (Multiply by $^2/_3$) 111 Reserved
15 DRDE	Delayed read discarded enable. Enables CPU interrupt generation when delayed read buffer is discarded because the request was not retried by the external PCI master in under 2^{15} PCI clock cycles. When enabled and the above condition occurs, software must clear PCIGSCR[DRD] to clear the interrupt condition. 0 Interrupt disabled 1 Interrupt enabled
14	Reserved, must be cleared.
13 PEE	PERR detected interrupt enable. Enables CPU interrupt generation when a PCI parity error is detected on the PCI_PERR signal. When enabled and PCI_PERR asserts, software must clear PCIGSCR[PE] to clear the interrupt condition. 0 Interrupt disabled 1 Interrupt enabled
12 SEE	SERR detected interrupt enable. Enables CPU interrupt generation when a PCI system error is detected on the PCI_SERR signal. When enabled and PCI_SERR asserts, software must clear PCIGSCR[SE] to clear the interrupt condition. 0 Interrupt disabled 1 Interrupt enabled
11 ERE	Error response interrupt enable. Enables CPU interrupt generation when an error response is detected on the internal crossbar switch bus when a PCI target transaction is attempted. When enabled and an internal error occurs, software must clear the PCIGSCR[ER] status bit to clear the interrupt condition. 0 Interrupt disabled 1 Interrupt enabled
10–1	Reserved, must be cleared.
0 PR	PCI reset. Controls the external PCI_RST signal. When this bit is cleared, external PCI_RST deasserts. Setting this bit when clear generates a strobe to some of the internal logic to return it to its reset value. It resets all PCI Type 0 configuration registers and resets the internal PCI_AD[31:0], PCI_PAR, and PCI_CBE[3:0] output lines. If grant to the PCI controller is asserted in the cycle after the reset strobe generates, the external PCI_AD[31:0], PCI_PAR, and PCI_CBE[3:0] pins are driven low. The application software must not initiate PCI transactions while this bit is set. It is recommended that this bit is programmed last during initialization. The reset value of the bit is 1 (PCI_RST asserted).

22.3.2.2 Target Base Address Translation Register 0 (PCITBATR0)

The next two registers, PCITBATR0 and PCITBATR1, are aliases for the registers at address 0xFC0A_8090 and 0xFC0A_8094 respectively. When these registers are written to, it also updates the contents of the other PCITBATR0 and PCITBATR1 registers. Likewise, when PCITBATR0 at address

0xFC0A_8090 is written to, it updates the contents of the register at address 0xFC0A_8064. When PCITBATR1 at address 0xFC0A_8094 is written to, it updates the contents of the register at address 0xFC0A_8068. This provides software compatibility with legacy devices using only the first two BARs (e.g. the MCF548*x* ColdFire processor).

Address: 0xFC0A_8064 (PCITBATR0) Access: User 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
R	BAT0														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-16. PCITBATR0 Register

Table 22-12. PCITBATR0 Field Descriptions

Field	Description
31–18 BAT0	Base address translation 0. Corresponds to a hit on the BAR0 in PCI Type 0 configuration space. When there is a hit on the PCI address indicated by PCI BAR0 (the ColdFire processor as target), the upper 14 bits of the address (256-Kbyte boundary) are written over by this register value to address some space in the device. In normal operation, this value must be written during the initialization sequence only.
17–1	Reserved, must be cleared.
0 EN	Enable 0. Enables a transaction in BAR0 space. If this bit is zero and a hit on the PCI address space indicated by PCIBAR0 occurs, the target interface gasket aborts the PCI transaction.

22.3.2.3 Target Base Address Translation Register 1 (PCITBATR1)

Address: 0xFC0A_8068 (PCITBATR1) Access: User 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
R	BAT1												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-17. PCITBATR1b Register

Table 22-13. PCITBATR1 Field Descriptions

Field	Description
31–20 BAT1	Base address translation 1. Corresponds to a hit on BAR1 in PCI Type 0 configuration space register (PCI space). When there is a hit on the PCI address indicated by PCI BAR1 (the ColdFire processor as target), the upper 12 bits of the address (1-Mbyte boundary) are written over by this register value to address some 1-Mbyte space in internal address space. This register can be reprogrammed to move the window of the device address space accessed during a hit in PCIBAR1. In normal operation, this value must be written during the initialization sequence only.
19–1	Reserved, must be cleared.
0 EN	Enable 1. Enables a transaction in BAR1 space. If this bit is zero and a hit on PCI address space indicated by BAR1 occurs, the target interface gasket aborts the PCI transaction.

22.3.2.4 Target Control 1 Register (PCITCR1)

Address: 0xFC0A_806C (PCITCR1)

Access: User 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
R	0	0	0	0	0	0	0	LD	0	0	0	0	0	0	PID	P	0	0	0	0	0	0	0	WCD	WCT							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-18. PCITCR1 Register

Table 22-14. PCITCR1 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 LD	Latency rule disable. Applies only when ColdFire processor is target. When set, it prevents the PCI controller from automatically issuing a retry disconnect due to the PCI 16/8 clock rule.Normal operation relies on the LD bit being cleared. If used, the bit must be set before the 15th PCI clock for first transfer and before the 7th clock for other transfers.
23–18	Reserved, must be cleared.
17 PID	Prefetch invalidate and disable. User sets this bit to invalidate all target prefetched data in the PCI controller. Software can use this bit to invalidate any stale prefetched data when internal memory has been modified by an internal resource. If prefetched data is currently transferring as read data to the PCI bus when this bit is set, data transfers stop and the target interface disconnects the current PCI transaction. If set when prefetch data is currently transferring from the internal bus, data for the current internal bus is marked invalid. This bit is sticky and completely disables prefetching from the internal bus for PCI when set. This applies to MEMORY READ MULTIPLE, as well as MEMORY READ and MEMORY READ LINE commands. This bit must be cleared for the prefetch buffers to work.
16 P	Prefetch reads. Controls fetching a line from memory in anticipation of request from the external master. The target interface continues to prefetch lines from memory as long as PCI_FRAME asserts and there is space to store the data in the target read buffer. **Note:** This bit only applies to PCI reads in the address range for BAR1–5 (prefetchable memory). **Note:** Prefetching performs in response to a PCI memory-read-multiple command even if this bit is cleared.
15–9	Reserved, must be cleared.
8 WCD	Write combine disable. Applies only when device is target. When set, it prevents PCI controller from automatically combining write data to be sent out on the internal bus as a burst. Instead, data transfers as soon as possible on the internal bus as single-beat transactions. **Note:** Better target write performance is achieved when this bit cleared.
7–0 WCT	Write combine timer. Contains the timer value, in PCI clocks, used when a partial burst is buffered in the target write data path and write data stops transfer to local memory from the external PCI device. Every time a sequential beat of write data stores in the buffer, the counter resets with this value. If partial burst data is buffered (activating the countdown counter) and this field is reprogrammed to a value less than the current counter value, the counter jumps down to the new WCT value. This way, software can force the write buffer to flush data to the internal bus more quickly than when the counter was initialized. The reset value of the write combine timer is 0x08. All 8 bits are programmable.

22.3.2.5 Initiator Window *n* Base/Translation Address Register (PCIIW*n*BTAR)

The following register figure describes the three initiator window base/translation address registers.

Address: 0xFC0A_8070 (PCIIW0BTAR)
0xFC0A_8074 (PCIIW1BTAR)
0xFC0A_8078 (PCIIW2BTAR)

Access: User read/write

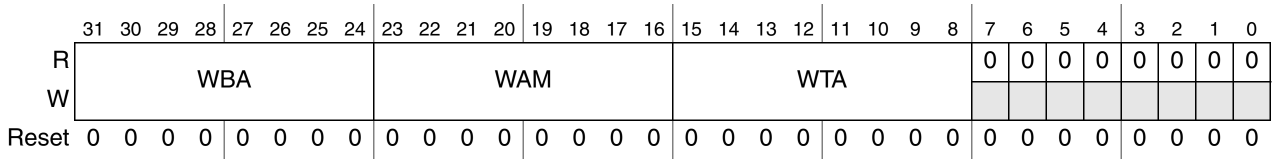


Figure 22-19. PCIIW*n*BTAR Register

Table 22-15. PCIIW*n*BTAR Field Descriptions

Field	Description
31–24 WBA	Window *n* base address. One of three base address fields to determine an internal bus hit on PCI. At most, the upper byte of the address is decoded. The WAM bit field determines what bits of this register to compare the internal bus address against to generate the hit. The smallest possible window is a 16-Mbyte block.
23–16 WAM	Window *n* address mask. Masks the corresponding internal bus base address bit of the base address for Window *n* (WBA) to instruct the address decode logic to ignore the bit. If the base address mask bit is set, the associated base address bit of window *n* is ignored when generating the PCI hit. Bit 16 masks bit 24, bit 17 masks bit 25, and so on. 0 Corresponding address bit used in address decode. 1 Corresponding address bit ignored in address decode. For internal bus accesses to the window *n* address range, this byte also determines which upper 8 bits of the internal bus address to pass on for presentation as a PCI address. Any address bit used to decode the internal bus address, indicated by a 0 translated. This provides a way to overlay a PCI page address onto the internal bus address. A 1 in the WAM byte indicates that the internal bus address bit is passed to PCI unaltered.
15–8 WTA	Window *n* translation address. For any translated bit (described above), the corresponding value here is driven onto the PCI address bus for the internal bus window 0 address hit. The window translation operation can not be turned off. If a direct mapping from internal bus to PCI space is desired, program the same value to the window base address register and window translation address register.
7–0	Reserved, must be cleared.

22.3.2.6 Initiator Window Configuration Register (PCIIWCR)

Address: 0xFC0A_8080 (PCIIWCR)

Access: User read/write

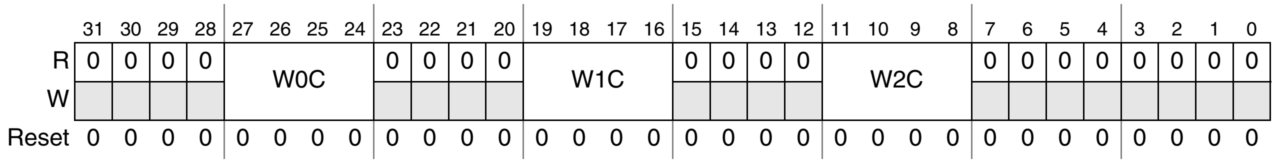
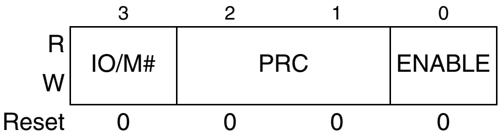


Figure 22-20. PCIIWCR Register

Table 22-16. PCIIWCR Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27–24 W0C	Window 0 control. The W0C, as well as the W1C and W2C bit fields contain the following bit structure.  3 2 1 0 R W IO/M# PRC ENABLE Reset 0 0 0 0 **Figure 22-21. Window *n* Control Bit Fields (W*n*C)** IO/M#: 0 Window maps to PCI memory. 1 Window maps to PCI I/O. PCI read command (PRC). If the corresponding IO/M# bit is cleared, these bits determine the type of PCI memory command to issue. If IO/M# is set, the value of these bits is ignored. 00 PCI MEMORY READ. 01 PCI MEMORY READ LINE. 10 PCI MEMORY READ MULTIPLE. 11 Reserved. Enable. Indicates the address registers controlling the internal bus initiator interface access to PCI for this window are initialized and used. The PCI controller can begin to decode internal bus PCI accesses. 0 Do not decode internal bus PCI accesses to window. 1 Registers initialized—decode accesses to window.
23–20	Reserved, must be cleared.
19–16 W1C	Window 1 control. See W0C bit for field description.
15–12	Reserved, must be cleared.
11–8 W2C	Window 2 control. See W0C bit for field description.
7–0	Reserved, must be cleared.

22.3.2.7 Initiator Control Register (PCIICR)

Address: 0xFC0A_8084 (PCIICR) Access: User 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
R	0	0	0	0	0	REE	IAE	TAE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Maximum Retries							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 22-22. PCIICR Register

Table 22-17. PCIICR Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26 REE	Retry error enable. Enables CPU interrupt generation in case of retry error termination of transaction. It may be desirable to mask CPU interrupts, but in such a case, software should poll the status bits to prevent a possible lock-up condition.
25 IAE	Initiator abort enable. Enables CPU interrupt generation in the case of initiator abort termination of a transaction.
24 TAE	Target abort enable. Enables CPU interrupt generation in the case of target abort termination of a transaction. It may be desirable to mask CPU interrupts, but in such a case, software should poll the status bits to prevent a possible lock-up condition.
23–8	Reserved, must be cleared.
7–0 Maximum Retries	Controls the maximum number of automatic PCI retries or master latency time-outs to permit per transaction. The retry counter is reset at the beginning of each transaction (i.e. it is not cumulative). Setting the maximum retries to 0x00 allows infinite automatic retry cycles and latency time-outs before the transaction aborts and, if open, sends back an error on the internal bus. A slow or malfunctioning target might issue infinite retry disconnects or hold the data tenure open indefinitely, and therefore, permanently tie up the PCI bus if no target abort occurs.

22.3.2.8 Initiator Status Register (PCIISR)

Address: 0xFC0A_8088 (PCIISR) Access: User 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
R	0	0	0	0	0	RE	IA	TA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W						w1c[1]	w1c[1]	w1c[1]																								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Bits 26-24 are write-one-to-clear (w1c).
Hardware can set w1c bits, but cannot clear them.
Software can clear w1c bits that are currently set by writing a 1 to the bit location. Writing a 1 to a w1c bit that is currently a 0 or writing a 0 to any w1c bit has no effect.

Figure 22-23. PCIISR Register

Table 22-18. PCIISR Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26 RE	Retry error. This flag is set when the Max_Retries limit is reached for the same internal bus transaction. A retry error would generally indicate a broken or improperly accessed target. A CPU interrupt generates if the PCIICR[RE] bit is set. In addition, when this error occurs while reaching the write retry limit, a read or a non-posted write causing a retry error generates an internal error. Writing a 1 to it clears this bit.
25 IA	Initiator abort. This flag bit is set if the PCI controller terminates a transaction with master abort during an initiator transaction. This indicates no target responded by asserting $\overline{\text{PCI\_DEVSEL}}$ within the time allowed for subtractive decoding. A CPU interrupt generates if the PCIICR[IAE] bit is set. Application software must write a 1 to it clears this bit.

Table 22-18. PCIISR Field Descriptions (continued)

Field	Description
24 TA	Target abort. This flag bit is set if the addressed PCI target has signalled an abort for an initiator transaction. A CPU interrupt generates if the PCIICR[TAE] bit is set. Application software must query the target's status register and determine the source of the error. In addition, an internal error generates when a read or non-posted write causes the target abort. Application software must write a 1 to it clears this bit.
23–0	Reserved, must be cleared.

22.3.2.9 Target Control 2 Register (PCITCR2)

Address: 0xFC0A_808C (PCITCR2)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	B5E	B43	B3E	B2E	B1E	B0E	0	0	0	0	0	0	0	CR
W																
Reset	0	0	—	—	—	—	—	—	0	0	0	0	0	0	0	—

Figure 22-24. PCITCR2 Register

Table 22-19. PCITCR2 Field Descriptions

Field	Description
31–14	Reserved, must be cleared.
13–8 B*n*E	BAR5 through BAR0 enable. Enables use of each target BAR register. Value of reset configuration pins determines the enable register reset value and number BAR registers implemented at reset. If programmed to 1, the respective BAR registers implement. If low, use of the BAR register is disabled. Initialization software can enable desired BAR registers any time before PCI enumeration.
7–1	Reserved, must be cleared.
0 CR	Configuration retry. Controls the response of the PCI controller to inbound accesses to allow additional time for internal configuration. If set, configuration accesses are retried. When cleared, the accesses complete. The level of the strapping pin, host/agent# mode, controls the reset value for this bit. If the host/agent# mode pin is high during hardware reset indicating host mode, reset value of this register is 0, allowing inbound accesses to complete normally. If low (agent mode), the bit is 1 out of reset, forcing configuration accesses to retry until initialization software clears this bit. **Note:** For normal target operation, this bit must be cleared. After cleared, this bit must not be set again unless resetting the PCI system.

22.3.2.10 Target Base Address Translation Register *n* (PCITBATR*n*)

Address: 0xFC0A_8090 (PCITBATR0) Access: User 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
R	BAT0														0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-25. PCITBATR0 Register

Address: 0xFC0A_8094 (PCITBATR1) Access: User 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
R	BAT1												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-26. PCITBATR1 Register

Address: 0xFC0A_8098 (PCITBATR2) Access: User 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
R	BAT2										0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-27. PCITBATR2 Register

Address: 0xFC0A_809C (PCITBATR3) Access: User 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
R	BAT3								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-28. PCITBATR3 Register

Address: 0xFC0A_80A0 (PCITBATR4) Access: User 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
R	BAT4					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-29. PCITBATR4 Register

Address: 0xFC0A_80A4 (PCITBATR5) Access: User 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
R	BAT5			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-30. PCITBATR5 Register

Table 22-20. PCITBATR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>See Figures[1]
BAT<i>n</i></td><td>Base address translation <i>n</i>. Corresponds to BAR<i>n</i> in the PCI Type 0 configuration space. When PCIBAR<i>n</i> (PCI controller as target) indicates a hit on the PCI address, the upper bits of the PCI address are written over by this register value to address some space in internal address space. See the below table for the number of PCI address bits overwritten and corresponding boundary size. In normal operation, this value must be written during the initialization sequence only.
<table>
<tr><th>BAT<i>n</i></th><th>Bit Field Width</th><th># of PCI Address Bits Overwritten</th><th>Boundary Size</th></tr>
<tr><td>BAT0</td><td>14</td><td>14</td><td>256 Kbye</td></tr>
<tr><td>BAT1</td><td>12</td><td>12</td><td>1 Mbyte</td></tr>
<tr><td>BAT2</td><td>10</td><td>10</td><td>4 Mbyte</td></tr>
<tr><td>BAT3</td><td>8</td><td>8</td><td>16 Mbyte</td></tr>
<tr><td>BAT4</td><td>5</td><td>5</td><td>128 Mbyte</td></tr>
<tr><td>BAT5</td><td>3</td><td>3</td><td>512 Mbyte</td></tr>
</table></td></tr>
<tr><td>See Figures[1]</td><td>Reserved, must be cleared.</td></tr>
<tr><td>0
EN</td><td>BAR0 enable. Enables a transaction in BAR0 space. If this bit is cleared and a hit on the PCI address space indicated by PCIBAR0 occurs, the target interface gasket targets abort the PCI transaction.</td></tr>
</table>

[1] See corresponding PCIBATR*n* figure above for bit numbers for the BAT*n* and reserved bit fields.

22.3.2.11 Interrupt Register (PCIINTR)

Address: 0xFC0A_80A8 (PCIINTR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	INT
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-31. PCIINTR Register

Table 22-21. PCIINTR Field Descriptions

Field	Description
31–1	Reserved, must be cleared.
0 INT	Interrupt. Controls the external $\overline{\text{PCI\_INTA}}$ signal. When this bit is set, the external active low $\overline{\text{PCI\_INTA}}$ signal is asserted. When this bit is cleared, $\overline{\text{PCI\_INTA}}$ is negated. Software must program this bit high when $\overline{\text{PCI\_INTA}}$ assertion is desired. The reset value of the bit is 0 ($\overline{\text{PCI\_INTA}}$ not asserted). **Note:** External $\overline{\text{PCI\_INTA}}$ pin specifies asserted when low and is an open-drain signal (high impedance when negated).

22.3.2.12 Configuration Address Register (PCICAR)

Address: 0xFC0A_80F8 (PCICAR) Access: User 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
R	E	0	0	0	0	0	0	0	Bus Number								Device Number					Function Number			DWORD						0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-32. PCICAR Register

Table 22-22. PCICAR Field Descriptions

Field	Description
31 E	Enable. Controls configuration space mapping. 0 Disabled. A read or write to the window passes through to the PCI bus as an I/O transaction. 1 Enabled. Subsequent access to initiator window space defined as I/O in the PCIIWCR register translates into a PCI configuration, special cycle, or interrupt acknowledge access using the configuration address register information (Section 22.4.3.2, “Configuration Mechanism”).
30–24	Reserved, must be cleared.
23–16 Bus Number	Bus number. Selects the target bus of the configuration access. For target devices on the PCI bus connected to this processor, this field must be set to 0x00.
15–11 Device Number	Device number. Selects a specific device on the target bus. See Section 22.4.3.2, “Configuration Mechanism,” for more information.
10–8 Function Number	Selects a specific function in the requested device. Single-function devices must respond to function number 000.
7–2 DWORD	DWord address offset. Selects the Dword address offset in the configuration space of the target device.
1–0	Reserved, must be cleared.

22.3.3 PCI Arbiter Registers

The PCI arbiter provides contains two registers accessed by 8-bit, 16-bit, or 32-bit accesses.

22.3.3.1 PCI Arbiter Control Register (PACR)

Address: 0xFC0A_C000 (PACR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	DS	PKMD	0	0	0	0	0	0	0	0	0	EXTMINTEN				INT MINTEN
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RA	0	0	0	0	0	0	0	0	0	0	EXT_MPRI				INT MPRI
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-33. PACR Register

Table 22-23. PACR Field Descriptions

Field	Description
31 DS	Disable. Disables the internal PCI arbiter. 0 Enable the PCI arbiter. 1 Disable the on-chip arbiter and use $\overline{\text{PCI\_GNT}}$[0] for the PCI controller request output and $\overline{\text{PCI\_REQ}}$[0] for its grant input.
30 PKMD	Parking mode. Controls which master takes bus ownership when no device uses or requests the bus. 0 Park with last master to use the bus. 1 Park with PCI controller.
29–21	Reserved, must be cleared.
20–17 EXTMINTEN	External master broken interrupt enables. If an external master time-out occurs and the corresponding interrupt enable bit is set, a CPU interrupt generates. Bit 20 is the enable for PASR bit 20, bit 19 for PASR bit 19, and so on. 0 Disable interrupt 1 Enable interrupt Software must write 1 to the corresponding PASR[EXTMBK] bit to clear the interrupt condition.
16 INTMINTEN	Internal master broken interrupt enable. If a PCI Controller master time-out occurs (PASR[ITLMBK]) and this bit is set, a CPU interrupt generates. 0 Disable interrupt 1 Enable interrupt Software must write 1 to the PASR[ITLMBK] bit to clear the interrupt condition.
15 RA	Reset arbiter. Puts the PCI arbiter in a reset state. Other PACR register bits are not affected, but all bits PASR register are cleared. If the PCI arbiter detects any broken masters when this bit is set, ignore condition clears. When this bit subsequently clears, requests from broken masters are once again recognized and arbitration resumes. This reset bit does not prohibit register access, but it must be cleared for arbitration to occur. When set, the arbiter parks with the internal master.
14–5	Reserved, must be cleared.

Table 22-23. PACR Field Descriptions (continued)

Field	Description
4–1 EXTMPRI	External master priority levels. Bit 1 controls the priority for the device using $\overline{\text{PCI\_REQ}}$[0] and $\overline{\text{PCI\_GNT}}$[0] pins, bit 2 for $\overline{\text{PCI\_REQ}}$[1] and $\overline{\text{PCI\_GNT}}$[1], etc. 0 Low 1 High
0 INTMPRI	Internal master priority level 0 Low 1 High

22.3.3.2 PCI Arbiter Status Register (PASR)

Address: 0xFC0A_C004 (PASR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	EXTMBK				ITLMBK	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W												w1c[1]				w1c[1]																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] Bits 20-16 are write-one-clear (w1c).
Hardware can set w1c bits, but cannot clear them.
Software can clear w1c bits that are currently set by writing a 1 to the bit location. Writing a 1 to a w1c bit that is currently a 0 or writing a 0 to any w1c bit has no effect.

Figure 22-34. PASR Register

Table 22-24. PASR Field Descriptions

Field	Description
31–21	Reserved, must be cleared.
20–17 EXTMBK	External master broken. Indicates an external master time-out has occurred. Bit 17 reports the time-out status for the device using $\overline{\text{PCI\_REQ}}$[0] and $\overline{\text{PCI\_GNT}}$[0] pins, bit 18 for $\overline{\text{PCI\_REQ}}$[1] and $\overline{\text{PCI\_GNT}}$[1], etc. A CPU interrupt generates if corresponding PACR[EXTMINTEN] bit is set. Software must write a 1 to each bit location or write a 1 to the software reset bit, PACR[RA], to clear.
16 ITLMBK	Internal master broken. Indicates a PCI controller master time-out has occurred. A CPU interrupt generates if PACR[INTMINTEN] bit is set. Software must write a 1 to this bit location or write a 1 to the software reset bit, PACR[RA], to clear.
15–0	Reserved, must be cleared.

22.4 Functional Description

Figure 22-1 shows the PCI controller provides master and target PCI bus interfaces. The internal master, or initiator, interface is accessible by any crossbar switch bus master, such as the processor core. The target interface provides external PCI masters access into as many as six memory windows of address space. PCI arbitration manages by the internal PCI arbiter or off-chip. (See Section 22.4.5, “PCI Arbiter”).

The registers, described in Section 22.3, “Memory Map/Register Definition,” control and provide information about multiple interfaces. An additional configuration interface allows internal access through the slave bus to the PCI Type 0 configuration registers accessible to internal and external masters through the PCI bus.

The following sections describe the operation of the PCI controller.

22.4.1 PCI Bus Protocol

This section provides a simple overview of the PCI bus protocol, including some details of the PCI controller implementation. For details regarding PCI bus operation, refer to the *PCI Local Bus Specification, Revision 2.2*.

22.4.1.1 PCI Bus Background

The PCI interface is synchronous and is best used for bursting data in large chunks. Its maximum bandwidth approaches 266 Megabytes per second for the 32-bit implementation running at 66 MHz. A system contains one device responsible for configuring all other devices on the bus upon reset. Each device has 256 bytes of configuration space defining individual requirements to the system controller. These registers are read and written through a CONFIGURATION ACCESS command. The master starts a PCI transfer and is directed toward a specific target. A provision is made for broadcasting to several targets through the SPECIAL command. Data transfers through the use of memory and I/O read and write commands.

Table 22-25. PCI Command Encodings

$\overline{\text{PCI\_CBE}}$[3:0]	Command Type
0000	INTERRUPT ACKNOWLEDGE
0001	SPECIAL CYCLE
0010	I/O READ
0011	I/O WRITE
0100	Reserved
0101	Reserved
0110	MEMORY READ
0111	MEMORY WRITE
1000	Reserved
1001	Reserved
1010	CONFIGURATION READ
1011	CONFIGURATION WRITE
1100	MEMORY READ MULTIPLE
1101	DUAL ADDRESS CYCLE
1110	MEMORY READ LINE
1111	MEMORY WRITE AND INVALIDATE

22.4.1.2 Basic Transfer Control

The basic PCI bus transfer mechanism is a burst. A burst is an address phase followed by one or more data phases. Fundamentally, all PCI data transfers are controlled by three signals: PCI_FRAME, PCI_IRDY, and PCI_TRDY. An initiator asserts PCI_FRAME begin a PCI bus transaction and negates PCI_FRAME to end a PCI bus transaction. An initiator negates PCI_IRDY to force wait cycles, while a target negates PCI_TRDY to force wait cycles.

The PCI bus is considered idle when PCI_FRAME and PCI_IRDY are negated. The first clock cycle in which PCI_FRAME asserts indicates the beginning of the address phase. The address and bus command code transfer in that first cycle. The next cycle begins the first of one or more data phases. Data transfers between initiator and target in each cycle that PCI_IRDY and PCI_TRDY assert. The initiator (by negating PCI_IRDY) or target (by negating PCI_TRDY) may insert wait cycles in a data phase.

After an initiator asserts PCI_IRDY, it cannot change PCI_IRDY or PCI_FRAME until the current data phase completes, regardless of the state of PCI_TRDY. After a target asserts PCI_TRDY or PCI_STOP, it cannot change PCI_DEVSEL, PCI_TRDY, or PCI_STOP until the current data phase completes. In simpler terms, after an initiator or target has committed to the data transfer, it cannot back out.

When the initiator intends to complete only one more data transfer (which can happen immediately after the address phase), PCI_FRAME negates and PCI_IRDY asserts (or kept asserted) indicating the initiator is ready. After the target indicates the final data transfer (by asserting PCI_TRDY), the PCI bus may return to the idle state (PCI_FRAME and PCI_IRDY are negated) unless a fast back-to-back transaction is in progress. In the case of a fast back-to-back transaction, an address phase immediately follows the last phase.

22.4.1.3 PCI Transactions

The figures in this section show the basic MEMORY READ and MEMORY WRITE command transactions.

Figure 22-35 shows a PCI burst read transaction (2-beat). The signal PCI_FRAME is driven low to initiate the transfer. Cycle 1 is the address phase with valid address information driven on the AD bus and a PCI command driven on the PCI_CBE bus. In cycle 2, the AD bus is in a turnaround cycle because of the read on a muxed bus. The byte enables, which are active low, are driven onto the PCI_CBE bus in this clock. Any combination of byte enables can be asserted (none may be asserted individually). A target responds to an address phase by driving the PCI_DEVSEL signal. The specification allows for four types of decode operations. The target can drive PCI_DEVSEL in 1, 2, or 3 clocks depending on whether the target is a fast, medium, or slow decode device, respectively. A single device can drive PCI_DEVSEL if no other agent responds by the fourth clock. This is called subtractive decoding in PCI terminology. The this PCI controller is a medium target decode device.

A valid transfer occurs when PCI_IRDY and PCI_TRDY are asserted. If either are negated during a data phase, it is considered a wait state. The target asserts a wait state in cycles 3 and 5 of Figure 22-35. A master indicates the final data phase is to occur by negating PCI_FRAME. In Figure 22-35, the target responds as a medium device, driving PCI_DEVSEL in cycle 3.

The final data phase occurs in cycle 6. Another agent cannot start an access until cycle 8. A provision in the specification allows the current master to start another transfer in cycle 7 when certain conditions apply. Refer to fast back-to-back transfers in the PCI specification for more details.

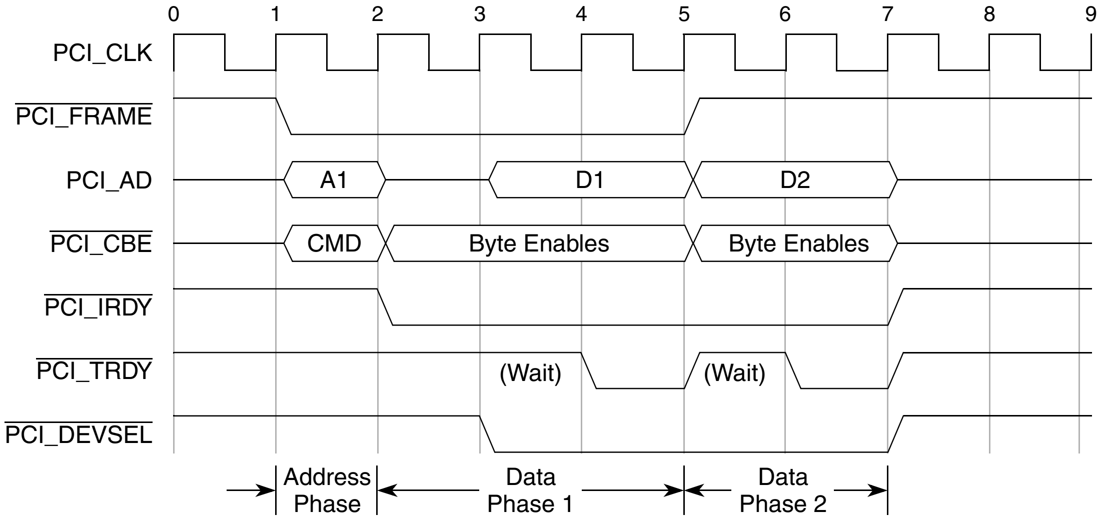


Figure 22-35. PCI Read Terminated by Master

Figure 22-36 shows a write cycle terminated by the target. In Figure 22-36, the target responds as a slow device, driving PCI_DEVSEL in cycle 4. The first data transfers in cycle 4. The master inserts a wait state at cycle 5. The target indicates it can accept only one more transfer by asserting PCI_TRDY and PCI_STOP at the same time in cycle 5. The signal PCI_STOP must remain asserted until PCI_FRAME negates. The final data phase does not have to transfer data. If PCI_STOP and PCI_IRDY are asserted while PCI_TRDY negates, it is considered a target disconnect without a transfer. See the PCI specification for more details.

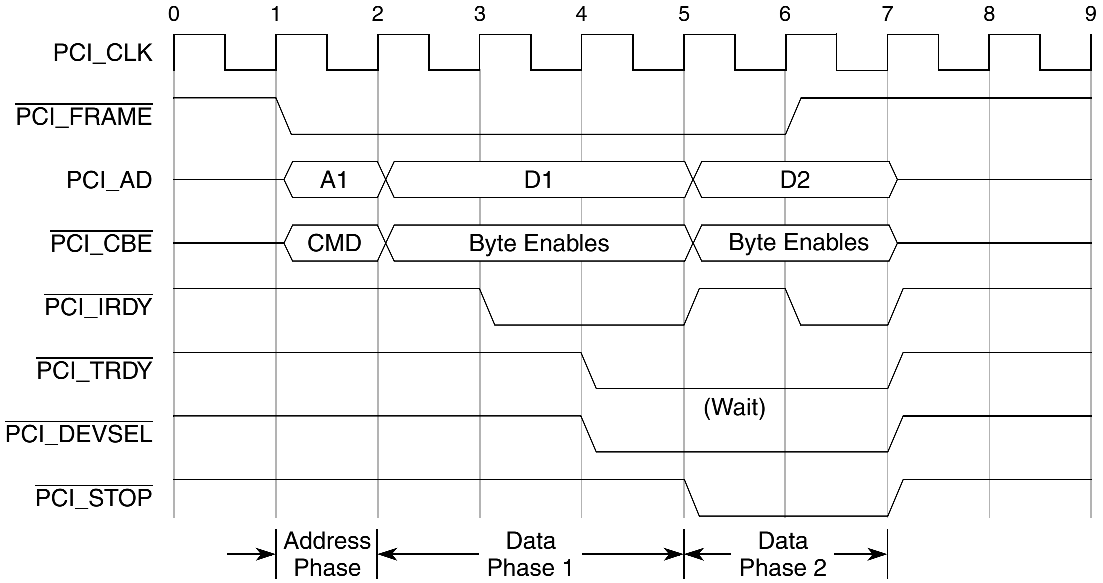


Figure 22-36. PCI Write Terminated by Target

22.4.1.4 PCI Bus Commands

PCI supports a number of different commands. The initiator on the PCI_CBE[3:0] signals during the address phase of a PCI transaction presents these commands.

Table 22-26. PCI Bus Commands

PCI_CBE[3:0]	PCI Bus Command	PCI Controller Supports as Initiator	PCI Controller Supports as Target	Definition
0000	INTERRUPT ACKNOWLEDGE	Yes	No	The INTERRUPT ACKNOWLEDGE command is a read (implicitly addressing an external interrupt controller). Only one device on the PCI bus should respond to the INTERRUPT ACKNOWLEDGE command.
0001	SPECIAL CYCLE	Yes	No	The SPECIAL CYCLE command provides a mechanism to broadcast select messages to all devices on the PCI bus.
0010	I/O READ	Yes	No	The I/O READ command accesses agents mapped into the PCI I/O space.
0011	I/O WRITE	Yes	No	The I/O WRITE command accesses agents mapped into the PCI I/O space.
0100	Reserved	No	No	—
0101	Reserved	No	No	—
0110	MEMORY-READ	Yes	Yes	The MEMORY READ command accesses agents mapped into PCI memory space.
0111	MEMORY-WRITE	Yes	Yes	The MEMORY WRITE command accesses agents mapped into PCI memory space.
1000	Reserved	No	No	—
1001	Reserved	No	No	—
1010	CONFIGURATION READ	Yes	Yes	The CONFIGURATION READ command accesses the 256 byte configuration space of a PCI agent.
1011	CONFIGURATION WRITE	Yes	Yes	The CONFIGURATION WRITE command accesses the 256 byte configuration space of a PCI agent.
1100	MEMORY READ MULTIPLE	Yes	Yes	For the PCI controller as master, the MEMORY READ MULTIPLE command functions the same as the MEMORY READ command. For the PCI controller as target, the MEMORY READ MULTIPLE command causes an internal bus burst and can prefetch an additional three bursts worth of data when addressed to prefetchable space and when PCITCR1[PID] clears. Internal bus prefetching only applies to PCI reads in the address range for BAR1–5. Cache line wrap implements if internal bus is the transaction initiator and also wraps.
1101	DUAL ADDRESS CYCLE	No	No	The DUAL ADDRESS CYCLE command transfers a 64-bit address (in two 32-bit address cycles) to 64-bit addressable devices. This device does not respond to this command.

Table 22-26. PCI Bus Commands (continued)

PCI_CBE[3:0]	PCI Bus Command	PCI Controller Supports as Initiator	PCI Controller Supports as Target	Definition
1110	MEMORY READ LINE	Yes	Yes	The MEMORY READ LINE command indicates an initiator requests the transfer of an entire cache line. For the PCI controller, the MEMORY READ LINE functions the same as the MEMORY READ command. Cache line wrap is not implemented.
1111	MEMORY WRITE AND INVALIDATE	Yes (DMA access only)	Yes	The MEMORY WRITE AND INVALIDATE command indicates an initiator transfers an entire cache line; if this data is in any cacheable memory, that cache line needs to be invalidated. The MEMORY WRITE AND INVALIDATE functions the same as the MEMORY WRITE command. Cache line wrap is not implemented.

Though this device supports many PCI commands as an initiator, the communication subsystem initiator interface should use PCI MEMORY READ and MEMORY WRITE commands.

22.4.1.5 Addressing

PCI defines three physical address spaces: PCI memory space, PCI I/O space, and PCI configuration space. Every device for every PCI transaction performs address decoding on the PCI bus. Each agent is responsible for decoding its own address. The PCI specification supports two types of address decoding: positive decoding and subtractive decoding (refer to Section 22.4.1.5.4, “Address Decoding”). The address space accessed depends primarily on the type of PCI command used.

22.4.1.5.1 Memory Space Addressing

For memory accesses, PCI defines two types of burst ordering controlled by the two low-order bits of the address: linear incrementing (AD[1:0] equals 0b00) and cache wrap mode (AD[1:0] equals 0b10). The other two AD[1:0] encodings (0b01 and 0b11) are reserved.

For linear incrementing mode, the memory address is encoded/decoded using PCI_AD[31:2]. Thereafter, the address increments by 4 bytes after each data phase completes until the transaction terminates or completes (a 4 byte data width per data phase is implied). The two low-order bits of the address are still included in all the parity calculations.

As an initiator, the PCI controller supports linear incrementing and cache wrap mode. When an internal bus burst transaction is wrapped for memory transactions, the cache wrap mode automatically generates. For zero-word-aligned bursts and single-beat transactions, the PCI controller drives AD[1:0] to 0b00.

As a target, PCI controller treats cache wrap mode as a reserved memory mode when the cache line size field is cleared, PCICR1[CLS]. The PCI controller returns the first beat of data and signals a disconnect without data on the second data phase. When the cache line size field is programmed to a non-zero value, cache wrap bursts break up and transfer linearly on the internal bus. Reads from the PCI controller, implemented as delayed reads, always disconnect at the cache line boundary. The PCI controller is not optimized for wrapping bursts.

22.4.1.5.2 I/O Space Addressing

For PCI I/O accesses, all 32 address signals provide an address with granularity of a single byte. After a target has claimed an I/O access, it must determine if it can complete the entire access as indicated by the byte enable signals. If all selected bytes are not in the address range of the target, the entire access cannot complete. In this case, target does not transfer any data and terminates the transaction with a target-abort.

Table 22-27. PCI I/O Space Byte Decoding

Access Size	PCI_AD[1:0]	PCI_CBE[3:0]	Data
8-bit	00	*xxx*0	AD[7:0]
	01	*xx*01	AD[15:8]
	10	*x*011	AD[23:16]
	11	0111	AD[31:24]
16-bit	00	*xxx*0	AD[15:0]
	01	*xx*01	AD[23:8]
	10	*x*011	AD[31:16]
24-bit	00	*xxx*0	AD[23:0]
	01	*xx*01	AD[31:8]
32-bit	00	*xxx*0	AD[31:0]

22.4.1.5.3 Configuration Space Addressing and Transactions

PCI supports two types of configuration accesses. Their primary difference is the format of the address on the PCI_AD[31:0] signals during the address phase. The two low-order bits of the address indicate the format used for the configuration address phase: type 0 (AD[1:0] equals 0b00) or type 1 (AD[1:0] equals 0b01). Both address formats identify a specific device and a specific configuration register for that device:

- Type 0 configuration accesses select a device on the local PCI bus. They do not propagate beyond the local PCI bus and are claimed by a local device or terminated with a master-abort.
- Type 1 configuration accesses target a device on a subordinate bus through a PCI-to-PCI bridge, see Figure 22-39. Type 1 accesses are ignored by all targets except PCI-to-PCI bridges that pass the configuration request to another PCI bus.

When the controller initiates a configuration access on the PCI bus, it places the configuration address information on the AD bus and the configuration command on the PCI_CBE[3:0] bus. Setting AD[1:0] to 0b00 during the address phase indicates a Type 0 configuration transaction. The bit pattern tells the community of devices on the PCI bus the bridge that owns the PCI bus has already performed the bus number comparison and verified the request targets a device on its bus. Figure 22-37 shows the contents of the AD bus during the address phase of the Type 0 configuration access.

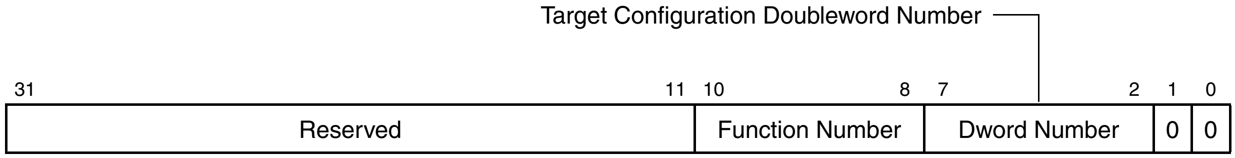


**Figure 22-37. Type 0 Configuration Transaction:
Contents of the AD Bus During Address Phase**

Address bits [10:8] identify the target function and bits AD[7:2] select one of the 64 configuration Dwords within the target function’s configuration space. For Type 0 configuration transactions, the target device’s IDSEL pin must be asserted. The upper 21 address lines are commonly IDSELs because they are not used during the address phase of a type 0 configuration transaction.

For a Type 1 access where the target bus is a subordinate bus to the local PCI bus (bus 0), the configuration transaction remains initiated on bus 0, but the bit pattern AD[1:0] indicates none of the devices on this bus are the target of the transaction. Rather, only PCI-to-PCI bridges on the local bus should pay attention to the transaction because it targets a device on a bus further out in the hierarchy beyond a PCI-to-PCI bridge that is attached to the local PCI bus (bus 0). Initiating a Type 1 configuration transaction (setting AD[1:0] to 0b01 during the address phase) accomplishes this. This pattern instructs all functions other than PCI-to-PCI bridges that the transaction is not for any of them. Figure 22-38 illustrates the contents of the AD bus during the address phase of the Type 1 configuration access.

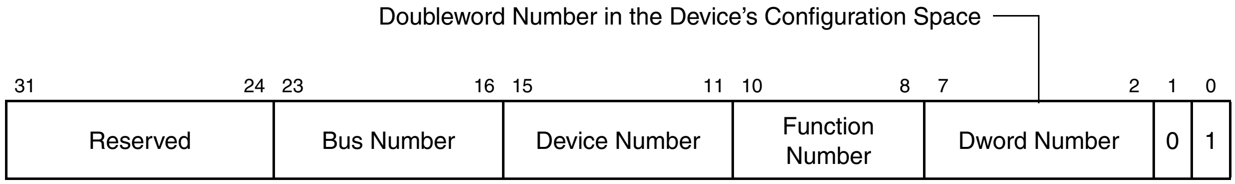


**Figure 22-38. Type 1 Configuration Transaction:
Contents of the AD Bus During Address Phase**

During the address phase of a Type 1 configuration access, the information on the AD bus is formatted as:

- PCI_AD[1:0] contains 0b01, identifying this as a Type 1 configuration access.
- PCI_AD[7:2] identifies one of 64 configuration Dwords within the target devices’s configuration space.
- PCI_AD[10:8] identifies one of the eight functions within the target physical device.
- PCI_AD[15:11] identifies one of 32 physical devices. The bridge uses this field to select which device’s IDSEL line to assert.
- PCI_AD[23:16] identifies one of 256 PCI buses in the system.
- PCI_AD[31:24] are reserved and are cleared to zero.

During a Type 1 configuration access, PCI devices ignore the state of their IDSEL inputs; PCI devices only respond to Type 0 accesses. When any PCI-to-PCI bridge latches a Type 1 configuration access (command equals CONFIGURATION READ or CONFIGURATION WRITE and AD[1:0] equals 0b01) on its primary side, it must determine whether the bus number field on the AD bus matches the number of its secondary bus or if the field is within the range of its subordinate buses. If the bus number matches, it should claim and pass the configuration access onto its secondary bus as a Type 0 configuration access, decoding the device number to select one of the IDSEL lines. If the bus number is not equal to its secondary bus, but is within the range of buses subordinate to the bridge, the bridge claims and passes that access through as a Type 1 access.

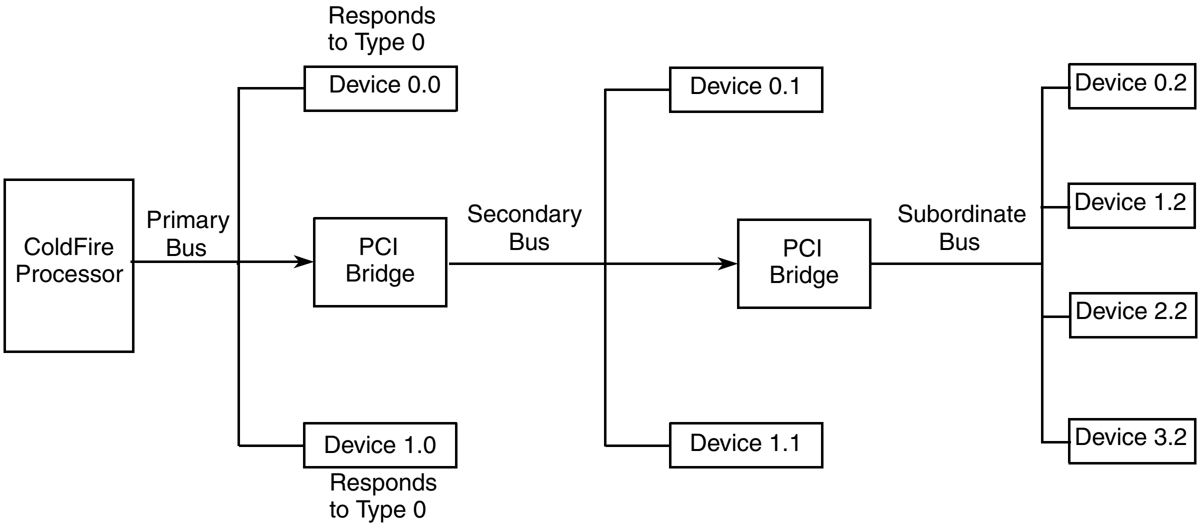


Figure 22-39. PCI-to-PCI Bridge Determining Match to Secondary or Subordinate Bus

22.4.1.5.4 Address Decoding

For positive address decoding, an address hits when the address on the address bus matches an assigned address range. Multiple devices on the same PCI bus may use positive address decoding, though no overlap in the assigned address ranges can occur. The PCI controller only implements positive address decoding.

For subtractive address decoding, an address hits when the address on the address bus does not match any address range for any of the PCI devices on the bus. Only one device on a PCI bus may use subtractive address decoding, and its use is optional.

22.4.2 Configuration Interface

The PCI bus protocol requires the implementation of a standardized set of registers for most devices on the PCI bus. The PCI controller implements a Type 0 Configuration register set or header. These registers, discussed in Section 22.3.1, “PCI Type 0 Configuration Registers,” are primarily intended to be read or written by the PCI configuring master at initialization time through the PCI bus. The PCI controller provides internal access to these registers through an internal bus interface. As with most of the PCI controller registers, they are accessible by software in the address space at offsets. Internal accesses to the Type 0 configuration header do not require PCI arbitration when they are accessed internally and are allowed to execute regardless of whether any write data is posted in the PCI Controller.

If the PCI controller is the host device or configuring master, the internal bus interface configures the PCI Controller. An external master would configure the PCI controller through the external PCI bus.

More information on the standard PCI configuration register is in the *PCI Local Bus Specification, Revision 2.2*.

22.4.3 Internal Bus Initiator Interface

The processor core or internal masters may access the PCI bus via the internal bus initiator interface. This internal interface is accessed through three windows in the processor address space set up by base address and base address mask registers (Section 22.3.2.5, "Initiator Window n Base/Translation Address Register (PCIIWnBTAR)"). Valid values programmed to the initiator window base address field in the PCIIW*n*BTAR registers include addresses within the crossbar switch slave port offset for the PCI controller. The controller does not receive and, therefore, does not decode addresses outside the crossbar switch slave port range. The base address registers must be enabled by setting their respective enable bits in the PCIIWCR register (Section 22.3.2.6, "Initiator Window Configuration Register (PCIIWCR)"). Accesses to this area translate into PCI transactions on the PCI bus. See Section 22.5.2, "Address Translation," for examples on setting up address windows. An internal address within the crossbar switch slave port range dedicated to PCI initiator space is not a hit on an PCIIW*n*BTAR[WBA] field results in an internal error.

The PCI configuration bits associated with the address window (PCIIWCR) determines the particular type of PCI transaction generated. For example, the user might set one window to do PCI MEMORY READ MULTIPLE accesses, one window for PCI I/O READ or I/O WRITE accesses, and the other window to do non-prefetchable (memory-mapped I/O) PCI memory accesses. See Table 22-16 for command translations.

In addition to configurable address window mapping logic, the register interface provides a configuration address register, which provides the ability to generate configuration, interrupt acknowledge, and special cycles. This interface configures external PCI devices. See Section 22.4.3.2, "Configuration Mechanism" for CONFIGURATION READ/WRITE, INTERRUPT ACKNOWLEDGE, and SPECIAL CYCLE command support.

The internal bus initiator interface supports all internal bus transactions, including single-beat transfers and bursts (16 bytes).

Internal bus initiator read requests are decoded into four types: PCI memory, I/O, configuration, and interrupt acknowledge. The PCI controller acknowledges the read address and attempts to gain access to the PCI bus to transfer read data. The address acknowledge may be delayed if posted write data moving in the other direction, the target interface path, posts when the previous read queues. This allows the target write data to complete to the internal bus before the PCI controller initiates a new PCI read transaction.

Internal bus initiator reads from PCI target space must obey PCI bridge ordering rules. The following sequence of events occur for every read across the initiator path:

1. Before transferring read data on the PCI bus, all previously posted writes moving across this interface complete first to the PCI bus.
2. The read performs on the PCI bus and queues in a buffer.
3. All posted writes moving in the other direction, across the internal bus target interface, and posted before the read occurred on the PCI bus complete to the internal bus.
4. The queued read data is then sent back to the internal bus master.

A 16-byte read buffer stores read data in the PCI controller when received from the PCI bus. PCI targets can disconnect in the middle of a transfer. If the target for an internal bus read from PCI disconnects part way through the burst, the PCI controller may have to manage a local memory access from an alternate

PCI master before the disconnected transfer can continue. The PCI controller continues to request mastership of the PCI bus until the original request completes.

For example, if the internal bus initiates a burst read and the PCI target disconnects after transferring the first half of the burst, the PCI controller re-arbitrates for the PCI bus; when granted, initiates a new transaction with the address of the third beat of the burst (4-beat internal bus bursts).

If the PCI read transaction completes normally, the PCI controller transfers back data to complete the internal read transaction. If the read results in an error, the retry limit is reached or target abort, an error response is sent to the internal master and no data transfers. If the read results in a master abort, all 1s are sent as data to the internal master.

When the PCI controller acts as an initiator/master, PCI critical-word-first (CWF) burst operation (i.e. cache line wrap burst) is supported, and the 2-bit cache line wrap address mode is driven on the address bus when the internal bus starts the burst at a non-zero-word-first address. This option provided only as a means for the initiator to support memory targets that support cache-line wrap. The processor is not permitted to cache from memory targets residing on the PCI bus.

Internal bus writes decode into PCI memory, PCI I/O, PCI configuration, or special cycles. If the transaction decodes into an I/O, configuration, or special cycle, the write connects. The PCI controller gains access to the PCI bus and successfully transfers the data before it asserts address acknowledge to the internal bus. If the address maps to PCI memory space, the internal bus address phase immediately acknowledges and write data posts in a buffer in the PCI controller.

A 16-byte write buffer posts memory writes from internal bus to PCI. Buffering minimizes the effect of the slower PCI bus on the higher-speed internal bus. It may contain single-beat internal bus write transactions or a single burst. After the internal bus write data latches internally, the internal bus is available for subsequent transactions without having to wait for the write to the PCI target to complete. If the write buffer is full and a subsequent internal bus write request to the PCI bus comes in, the data transfer delays until enough previous writes to the PCI bus complete to allow another write to post. Only when the write buffer is empty can burst data from the internal bus be posted.

22.4.3.1 Endian Translation

The PCI bus is inherently little endian in its byte ordering. The internal bus, however, is big endian. Internal bus transactions are limited to 1, 2, 4, or 16 byte (burst) transactions within the data bus byte lanes on any 32-bit address boundary for burst transfers. Table 22-28 shows the byte lane mapping between the two buses.

Table 22-28. Internal bus to PCI Byte Lanes for Memory[1] Transactions

Internal Bus						PCI Bus					
HADDR [1:0]	HSIZE [2:0]	Data Bus Byte Lanes				AD [1:0]	BE [3:0]	Data Bus Byte Lanes			
		31:24	23:16	15:8	7:0			31:24	23:16	15:8	7:0
00	001	OP7	—	—	—	00	1110	—	—	—	OP7
01	001	—	OP7	—	—	00	1101	—	—	OP7	—
10	001	—	—	OP7	—	00	1011	—	OP7	—	—

Table 22-28. Internal bus to PCI Byte Lanes for Memory[1] Transactions (continued)

Internal Bus						PCI Bus					
HADDR [1:0]	HSIZE [2:0]	Data Bus Byte Lanes				AD [1:0]	BE [3:0]	Data Bus Byte Lanes			
		31:24	23:16	15:8	7:0			31:24	23:16	15:8	7:0
11	001	—	—	—	OP7	00	0111	OP7	—	—	—
00	010	OP6	OP7	—	—	00	1100	—	—	OP7	OP6
10	010	—	—	OP6	OP7	00	0011	OP7	OP6	—	—
00	100	OP4	OP5	OP6	OP7	00	0000	OP7	OP6	OP5	OP4

[1] The byte lane translation is similar for other types of transactions. However, the PCI address may differ from Section 22.4.1.5, "Addressing."

22.4.3.2 Configuration Mechanism

To support Type 0 and Type 1 configuration transactions, the PCI controller provides the 32 bit configuration address register (PCICAR). The register specifies the target PCI bus, device, function, and configuration register accessed. A read or a write to the PCI controller window defined as PCI I/O space in the PCIIWCR causes the host bridge to translate the access into a PCI configuration cycle if the PCICAR[E] bit is set and the device number does not equal 0b1_1111. For space defined as I/O space, the accessed space (one of the initiator windows) must be programmed as I/O, not memory. See Section 22.3.2.6, "Initiator Window Configuration Register (PCIIWCR)."

Section 22.3.2.12, "Configuration Address Register (PCICAR)" shows the format of the PCICAR register. When the PCI controller detects an access to an I/O window, it checks the PCICAR[E,Device Number] fields. If the enable bit is set, and the device number is not 0b1_1111, the PCI controller performs a configuration cycle translation function and runs a CONFIGURATION READ or CONFIGURATION WRITE transaction on the PCI bus. The device number 0b1_1111 performs INTERRUPT ACKNOWLEDGE and SPECIAL CYCLE transactions. See Section 22.4.3.3, "Interrupt Acknowledge Transactions," and Section 22.4.3.4, "Special Cycle Transactions," for more information. If the bus number corresponds to the local PCI bus (bus number equals 0x00), a Type 0 configuration cycle transaction performs. If the bus number indicates a remote PCI bus, PCI controller performs a Type 1 configuration cycle translation. If the enable bit is not set, access to the configuration window passes through to the PCI bus as an I/O transfer (window translation applies).

NOTE

Size access to the window determines the PCI data byte enables (PCI_$\overline{\text{CBE}}$[3:0]).

22.4.3.2.1 Type 0 Configuration Translation

Figure 22-40 shows the Type 0 translation function performed on the contents of the configuration address register to the PCI_AD[31:0] signals on the PCI bus during the address phase of the configuration cycle (this only applies when the PCICAR[E] bit is set).

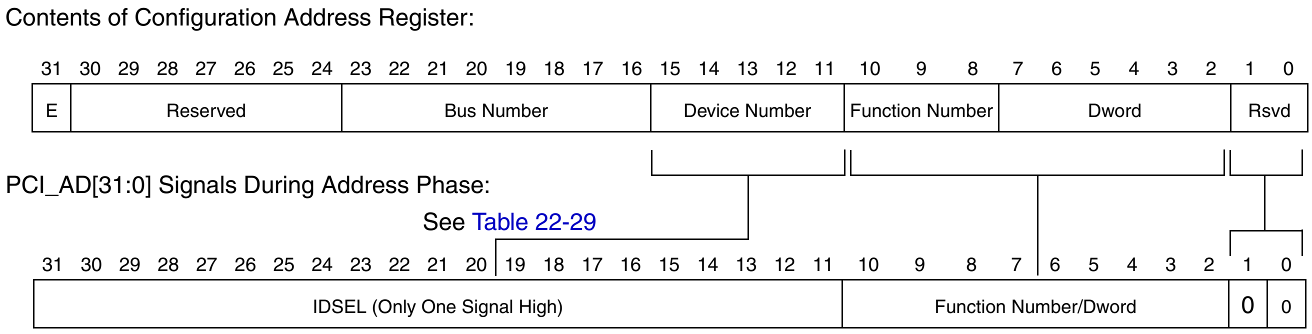


Figure 22-40. Type 0 Configuration Translation

For Type 0 configuration cycles, the PCI controller translates the device number field of the PCICAR register into a unique IDSEL line shown in Table 22-29. It allows for 21 different devices.

Table 22-29. Type 0 Configuration Device Number to IDSEL Translation

Device Number		IDSEL
Binary	Decimal	
0_0000 – 0_1001[1]	0 – 9	—
0_1010	10	AD31
0_1011	11	AD11
0_1100	12	AD12
0_1101	13	AD13
0_1110	14	AD14
0_1111	15	AD15
1_0000	16	AD16
1_0001	17	AD17
1_0010	18	AD18
1_0011	19	AD19
1_0100	20	AD20
1_0101	21	AD21
1_0110	22	AD22
1_0111	23	AD23
1_1000	24	AD24
1_1001	25	AD25
1_1010	26	AD26
1_1011	27	AD27
1_1100	28	AD28
1_1101	29	AD29

Table 22-29. Type 0 Configuration Device Number to IDSEL Translation (continued)

Device Number		IDSEL
Binary	Decimal	
1_1110	30	AD30
1_1111	31	—

[1] Device numbers 0b0_0000 to 0b0_1001 are reserved. Programming to these values and issuing a configuration transaction results in a PCI configuration cycle with AD31-AD11 driven low.

The PCI controller can issue PCI configuration transactions to itself. A Type 0 configuration initiated by the PCI controller can access its own configuration space by asserting its IDSEL input signal.

NOTE

> Asserting IDSEL is the only way the PCI controller can clear its own status register (PCISCR) bits (write-one-to-clear).

For Type 0 translations, the function number and Dword fields are copied without modification onto the PCI_AD[10:2] signals, and PCI_AD[1:0] are driven low during the address phase.

22.4.3.2.2 Type 1 Configuration Translation

For Type 1 translations, the 30 high-order bits of the PCICAR register are copied without modification onto the PCI_AD[31:2] signals during the address phase. The PCI_AD[1:0] signals are driven to 0b01 during the address phase to indicate a Type 1 configuration cycle.

22.4.3.3 Interrupt Acknowledge Transactions

When the PCI controller detects a read from an I/O defined window (Section 22.3.2.6, "Initiator Window Configuration Register (PCIIWCR)"), it checks the enable flag, bus number, and the device number in the PCICAR register (Section 22.3.2.12, "Configuration Address Register (PCICAR)"). If the PCICAR[E] bit is set, the bus number corresponds to the local PCI bus (bus number equals 0x00), and the device number is all 1's (device number equals 0b1_1111), an INTERRUPT ACKNOWLEDGE transaction is initiated. If the bus number indicates a subordinate PCI bus (bus number ! equals 0x00), a Type 1 configuration cycle is initiated, similar to any other configuration cycle the bus number does not match. The function number and Dword values are ignored.

The INTERRUPT ACKNOWLEDGE command (0b0000) is driven on the $\overline{\text{PCI\_CBE}}$[3:0] signals and the address bus is driven with a stable pattern during the address phase, but a valid address is not driven. The address of the target device during an INTERRUPT ACKNOWLEDGE is implicit in the command type. Only the system interrupt controller on the PCI bus should respond to the INTERRUPT ACKNOWLEDGE and return the interrupt vector on the data bus during the data phase. The size of the interrupt vector returned is indicated by the value driven on the $\overline{\text{PCI\_CBE}}$[3:0] signals.

22.4.3.4 Special Cycle Transactions

When the PCI controller detects a write to an I/O defined window (Section 22.3.2.6, “Initiator Window Configuration Register (PCIIWCR)”), it checks the enable flag, bus number, and the device number in the PCICAR register (Section 22.3.2.12, “Configuration Address Register (PCICAR)”). If the PCICAR[E] bit is set, the bus number corresponds to the local PCI bus (bus number equals 0x00), the device number is all 1s (device number equals 0b1_1111), a SPECIAL CYCLE transaction initiates. If the bus number indicates a subordinate PCI bus (bus number ! equals 0x00), a Type 1 configuration cycle is initiated, similar to any other configuration cycle the bus number does not match. The function number and Dword values are ignored.

The SPECIAL CYCLE command (0b0001) is driven on the $\overline{\text{PCI\_CBE}}$[3:0] signals and the address bus is driven with a stable pattern during the address phase, but contains no valid address information. The SPECIAL CYCLE command contains no explicit destination address, but broadcasts to all agents on the same bus segment. Each receiving agent must determine whether the message is applicable to it. PCI agent never asserts $\overline{\text{PCI\_DEVSEL}}$ in response to a SPECIAL CYCLE command. Master abort is the normal termination for a SPECIAL CYCLE and no errors report for this case of master abort termination. This command is basically a broadcast to all agents, and interested agents accept the command to process the request.

NOTE

> SPECIAL CYCLE commands do not cross PCI-to-PCI bridges. If a master wants to generate a SPECIAL CYCLE command on a specific bus in the hierarchy not its local bus, it must use a Type 1 CONFIGURATION WRITE command to do so. Type 1 CONFIGURATION WRITE commands can traverse PCI-to-PCI bridges in both directions for the purpose of generating SPECIAL CYCLE commands on any bus in the hierarchy and are restricted to a single data phase in length. However, the master must know the specific bus it desires to generate the SPECIAL CYCLE command on and cannot simply do a broadcast to one bus to expect it to propagate to all buses.

During the data phase, PCI_AD[31:0] contains the SPECIAL CYCLE message and an optional data field. The SPECIAL CYCLE message encodes on the 16 least significant bits (PCI_AD[15:0]) and the optional data field encodes on the most significant bits (PCI_AD[31:16]). The PCI SIG steering committee assign the SPECIAL CYCLE message encodings. Table 22-30 provides the current list of defined encodings.

Table 22-30. Special Cycle Message Encodings

PCI_AD[15:0]	Message
0x0000	SHUTDOWN
0x0001	HALT
0x0002	x86 architecture-specific
0x0003 – 0xFFFF	—

22.4.3.5 Transaction Termination

If the PCI cycle master aborts, interface returns 0xFFFF_FFFF as read data, but completes without error. It issues an interrupt to the internal interrupt controller if enabled.

For abnormal transaction termination during an internal bus-initiated transaction (unsupported transfer types, retry limit reached, or target abort) or if the crossbar switch transaction does not hit in the PCI-defined space, an error (bus error reported in the SCMISR, see Chapter 14, "System Control Module (SCM)," for more information) generates. It issues an interrupt to the processor's interrupt controller if such interrupts are enabled.

Internal bus burst transfers to a PCI non-memory address range result in a transfer error (bus error reported in the SCMISR). The space is non-memory if the IO/M# configuration bit associated with that window clears. This type of unsupported transfer does not cause a PCI interrupt, but can trigger an interrupt from the SCM.

Table 22-31. Unsupported Internal Bus Transfers

Internal Bus Transaction	PCI Address Space
Burst (16-byte)	Non-memory

22.4.4 Internal Bus Target Interface

This section discusses the PCI controller as a PCI target, and as such, the following apply:

- The target interface can issue target abort, target retry, and target disconnect terminations.
- The target interface supports fast back-to-back cycles.
- No support of dual address cycles as a PCI target.
- The processor does not snoop target transactions.
- Medium device selection timing only.
- Six 16-byte buffers enhance data throughput.

The internal bus target interface provides access for external PCI masters to as many as six windows of internal processor address space. The PCIBATR0–5 registers allow the user to map PCI address hits on the PCI controller's base address registers (PCIBAR*n*) to areas in the internal address space. At least one of these registers must be enabled for this interface to operate.

Upon detection of a PCI address phase, the PCI controller decodes the address and bus command to determine if transaction is for local memory (BAR hit). If transaction falls within the device's PCI space (memory only), PCI controller target interface asserts $\overline{\text{PCI\_DEVSEL}}$, latches the address, decodes the PCI bus command, and forwards them to the internal control unit as a privileged access. On writes, data forwards along with the byte enables to the internal control unit. On reads, four bytes of data are provided to the PCI bus and the byte enables determine which byte lanes contain meaningful data. If no byte enables asserts, PCI controller completes a read access with valid data and completes a write access by discarding the data internally. All other target memory transactions translate into internal bus master transactions.

There are address translation registers for each unique PCIBAR*n* register initialized before data transfer can begin. These address registers correspond to BAR0–5 in PCI Type 00h configuration space register

(PCI space). When a hit on PCI base address ranges (0 through 5), the upper bits of the address are written over by this register value to address some space in the processor. One 256-Kbyte base address range (BAR0) maps to non-prefetchable local memory and the rest (BAR1–5) target to prefetchable memory. Prefetching for PCI reads performs if the PCITCR1[PID] bit clears and the PCI command is a MEMORY READ MULTIPLE or the PCITCR1[P] bit sets and the read address falls in the range of prefetchable memory space.

22.4.4.1 Reads from Local Memory

The PCI controller implements delayed reads when accessed as a target device. The following sequence of events occur for every delayed read:

1. The read attempt, if valid, latches as a delayed request.
2. Before performing the read on the internalbus, all previously posted writes moving across the target interface complete first on the internal bus.
3. The read performs on the internal bus and queues in a target read buffer.
4. All posted writes moving in the initiator direction and posted before the read occurs on the internal bus complete to the external PCI target.
5. The queued target read data permits to be sent back to the external PCI master when re-attempted. The external master requires to re-attempt the target read.

After a delayed read completes on the internal bus, it may be discarded due to a tardy master. If the master that initiated the transaction does not retry the transaction in 2^{15} PCI clocks, delayed read discards and the PCIGSCR[DRD] status bit sets.

For prefetchable memory (BAR1–5 space), internal bus can fetch extra data to increase target read performance. Prefetching performs for BAR addressed transactions if the PCITCR1[PID] bit clears and the PCI command is a MEMORY READ MULTIPLE or the PCITCR1[P] bit is set. When a delayed request latches and prefetching is enabled, up to 16-bytes are fetched, if possible. The first 4 bytes is the delayed request and the next 28 bytes are prefetched. This data is stored in the first cacheline size read buffer. If no new target writes are transferred and no internal bus initiated memory writes to the PCI bus get posted, memory can fetch the next 3 lines in anticipation of the next PCI request and store in the second cacheline size read buffer. All prefetch data is invalidated when memory writes post in either direction across the PCI controller or when the PCITCR1[PID] is set. The prefetch data is any data outside the initial 4-byte PCI read requested by the external master and latched as a delayed request.

If a subsequent beat of a PCI read burst is requested and not stored in a read buffer, delayed or prefetched, the subsequent data phase disconnects without data transfer and is not treated as a delayed request. The external PCI master, if prefetching, may choose not to repeat the subsequent data phase as a new request and discard.

When an internal error (bus error response to the SCM) occurs on a read in delayed mode on the internal bus, it is not directly sent back to the PCI bus. It is stored in an interim register sent back when the external PCI master re-attempts the read. A target abort is issued to the PCI bus if the error occurred on the delayed read request. If prefetching is enabled and an error occurs on the prefetched access on the cache line of the delayed read access, it only generates a target abort if the error occurs prior to when the delayed read is sent to the external PCI master. Otherwise, the PCI controller does not store the prefetched data. If the next cache line is fetched internally and only this line returns an error, no data is stored in the PCI controller. No target abort issues on the PCI bus unless that cache line is requested as a delayed read.

If no byte enables are asserted, the PCI controller completes a read access with all four bytes of valid data.

22.4.4.2 Local Memory Writes

The target interface always posts writes. This allows for data to be latched while waiting for internal access to local memory. Two 16-byte posted write buffer are implemented to improve data throughput. When write buffers fill, the target interface keeps the PCI write transaction open on the PCI bus until the buffers empty enough to transfer more write data or the PCI 16/8 time-out is reached. When write buffers are full, the target interface negates $\overline{\text{PCI\_TRDY}}$ to insert data wait states. If write buffers do not empty some before 16 PCI clocks for the first write beat or 8 PCI clocks for subsequent beats and the PCITCR1[LD] bit is set, a disconnect issues for the PCI write transaction.

The first buffer simply posts data from the PCI bus and allows for conversion to the faster clock domain. The second buffer combines data for a potential internal bus burst. When data from the PCI bus aligns to a 16-byte burst and all byte enables transfer, it is stored in the second buffer. If subsequent PCI beats can combine with this stored data for a linear burst on the internal bus, all 16-bytes of data store in this buffer and sent out on the internal bus as a complete 16-byte burst. PCI write data must transfer with all byte enables enabled (32-bit) to be buffered for a burst on the internal bus, but can be combined from multiple consecutive PCI write transactions to complete the internal burst. If a target write to a non-sequential address or a target read occurs before an entire 32-byte burst is buffered, the write data stored transfers on the internal bus in single beat increments (64 bit, if possible). If partial burst write data (not a complete 16-byte burst) is stored and no new data comes from the PCI bus in a programmed number of PCI cycles, the buffer times out and sends in single beat. The PCITCR[WCT] field is available to program a timer value up to 255 PCI cycles. Setting the PCITCR[WCD] bit can also disable the write combine function. When this bit is set, the second 16-byte buffer is not used. The write combine function does not include byte packing.

If the PCI controller aborts the transaction in the middle of PCI burst due to internal conflicts, the external master recognizes some of the data before the abort transfers. However, subsequent transfers of a burst abort. The external PCI master must query the target abort signalled bit in the PCI Type 00h configuration status register to determine if a target abort occurred.

22.4.4.3 Data Translation

The internal bus does not support misaligned operations; therefore, it is recommended that software attempts to transfer contiguous code and data where possible. Non-contiguous transfers degrade performance. Table 22-32 and Table 22-33 show PCI-to-internal bus transaction data translation.

Table 22-32. Contiguous PCI to Internal bus Transfers

PCI Bus						Internal Bus				
$\overline{BE}$[3:0]	AD[1:0]	Data Bus Byte Lanes				HADDR [1:0]	Data Bus Byte Lanes			
		31:24	23:16	15:8	7:0		31:24	23:16	15:8	7:0
1110	00				OP3	00	OP3			
1101	00			OP3		01		OP3		
1011	00		OP3			10			OP3	
0111	00	OP3				11				OP3
1100	00			OP3	OP2	00	OP2	OP3		
1001	00		OP3	OP2		01		OP2		
						10			OP3	
0011	00	OP3	OP2			10			OP2	OP3
1000	00		OP3	OP2	OP1	00	OP1	OP2		
						10			OP3	
0001	00	OP3	OP2	OP1		01		OP1		
						10			OP2	OP3
0000	00	OP3	OP2	OP1	OP0	00	OP0	OP1	OP2	OP3

Table 22-33. Non-Contiguous PCI to Internal Bus Transfers (All Require Two Internal Bus Accesses)

PCI Bus						Internal Bus				
$\overline{BE}$[3:0]	AD[1:0]	Data Bus Byte Lanes				HADDR [1:0]	Data Bus Byte Lanes			
		31:24	23:16	15:8	7:0		31:24	23:16	15:8	7:0
1010	00		OP3		OP2	00	OP2			
						10			OP3	
0110	00	OP3			OP2	000	OP2			
						011				OP3
0101	00	OP3		OP2		01		OP2		
						11				OP3
0010	00	OP3	OP2		OP1	00	OP1			
						10			OP2	OP3
0100	00	OP3		OP2	OP1	00	OP1	OP2		
						11				OP3

22.4.4.4 Target Abort

A target abort occurs if PCI address falls within a base address window (BAR0–5) whose corresponding PCIBTAR*n* is not enabled. See Section 22.3.2.10, “Target Base Address Translation Register n (PCITBATRn).” Target aborts also issue on the PCI bus when a delayed read results in an error in the internal system.

22.4.4.5 Latency Rule Disable

The latency rule disable bit in the interface control register, PCITCR1[LD], (see Section 22.3.2.4, “Target Control 1 Register (PCITCR1)”) prevents the PCI controller from automatically disconnecting a target transaction due to the PCI 16/8 clock rule. With this bit set, it is possible to hang the PCI bus if the internal bus does not complete the data transfer.

22.4.5 PCI Arbiter

Out of reset, the PCI arbiter is disabled. If the application wants to enable the use of the internal PCI arbiter, initialization software must clear the PACR[DS] bit before PCI bus operation begins. The following sections detail the PCI arbiter functionality if enabled.

22.4.5.1 External PCI Requests

An external PCI master may target this device or external slaves. The request/grant handshake always precedes any PCI bus operation. The PCI arbiter must service access requests for an external master-to-external target transactions as well as external master-to-PCI controller transactions.

22.4.5.2 Hidden Bus Arbitration

PCI bus arbitration can take place while the currently granted device performs a bus transaction if another master requests access to the bus. As long as the bus is active, the arbiter can deassert $\overline{\text{PCI\_GNT}}$ to one master and assert $\overline{\text{PCI\_GNT}}$ to the next in the same cycle and no PCI bus cycles are consumed due to arbitration. The newly granted device must wait until the current master relinquishes the bus before initiating a transaction.

22.4.5.3 Arbitration Scheme

The PCI bus arbiter logic provides a programmable two-level least recently used (LRU) priority algorithm. Two groups of masters are assigned, a high-priority group and a low-priority group. The low-priority group as a whole represents one entry in the high-priority group. If the high-priority group consists of *n* masters in at least every *n*+1 transactions, the highest priority is assigned to the low-priority group. Low-priority masters have equal access to the bus with respect to other low-priority masters. If there are masters programmed into both groups, masters in the high-priority group can be serviced *n* transactions out of *n*+1 while one master in the low-priority group is serviced once every *n*+1 transactions. If all masters are programmed to the same group and only one master is assigned to the low-priority group, no priority distinction among masters exists.

A LRU priority scheme allows for fairness in priority resolution because no one master can prevent other masters from gaining access to the bus. The priority level, high or low, provides a simple weighting mechanism for master access to the bus.

Priority in a LRU scheme adjusts so that the last master serviced assigns the lowest priority in its level. Masters with lower priority shift to the next higher priority position. The PCI controller is positioned before all external devices in priority. If a master does not requesting the bus, its transaction slot is given to the next requesting device within its priority group.

During hidden arbitration, $\overline{\text{PCI\_GNT}}$ given to a requesting master while the PCI bus is active may be removed and awarded to a higher priority device if a higher priority device asserts its request. If the bus is idle when a device requests the bus, the arbiter deasserts the currently asserted $\overline{\text{PCI\_GNT}}$ for one PCI clock cycle. The arbiter evaluates the priorities of all requesting devices and grants the bus to the highest priority device in the next cycle.

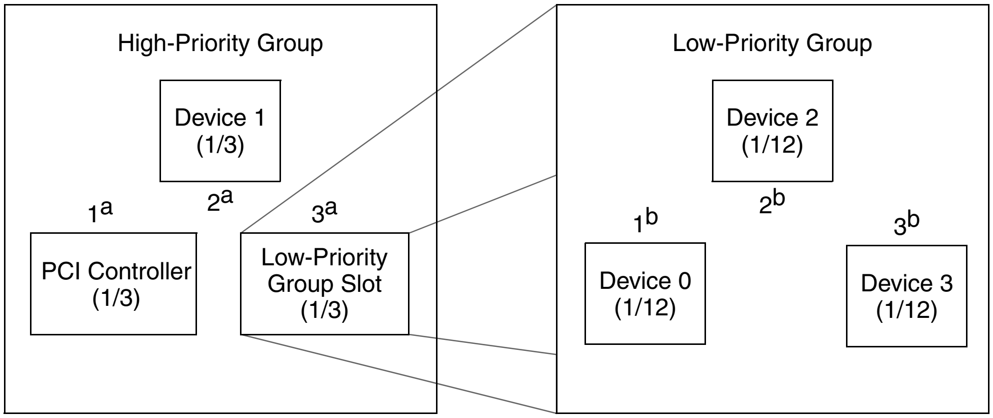
Figure 22-41 shows the initial state of the arbitration algorithm. Two devices are assigned high-priority (the internal and one external master) and four low-priority. If all masters request the use of the PCI bus continuously, the $\overline{\text{PCI\_GNT}}$ sequence is:

1. PCI controller, device 1, device 0,
2. PCI controller, device 1, device 2,
3. PCI controller, device 1, device 3,
4. Goto step 1.

If device 1 is not requesting the bus, the $\overline{\text{PCI\_GNT}}$ sequence is:

1. PCI controller, device 0,
2. PCI controller, device 2,
3. PCI controller, device 3,
4. Goto step 1.

If, after this sequence completes, all devices request the bus (including now device 1), the arbiter assigns $\overline{\text{PCI\_GNT}}$ to device 1 because it is the longest since device 1 used the bus (it has highest priority). After all requests are serviced, the priority resets to the initial state.



PCI Arbiter Control Register PACR[4:0] = 0_0101

Figure 22-41. PCI Arbitration Initial State

22.4.5.4 Arbitration Latency

Arbitration latency is the number of clock cycles from a master's $\overline{\text{PCI\_REQ}}$ assertion to PCI bus idle state and its $\overline{\text{PCI\_GNT}}$ assertion. In a lightly loaded system, arbitration latency would be the time it takes for the bus arbiter to assert the master's $\overline{\text{PCI\_GNT}}$ (zero cycles if the arbiter is parked with the requesting master and two if parked with another master). If a transaction is in progress when the master's $\overline{\text{PCI\_GNT}}$ asserts, the master must wait for the current transaction to complete and any subsequent transactions from higher priority requesting masters. In a situation where multiple requesting masters exist, each master latency timer limits the master's tenure on the bus.

22.4.5.5 Arbitration Examples

Figure 22-42 shows basic arbitration. Three master devices illustrate how an arbiter may alternate bus accesses. (Assume device 0, device 1, and device 2 are assigned the same priority group and no other masters request use of the bus.)

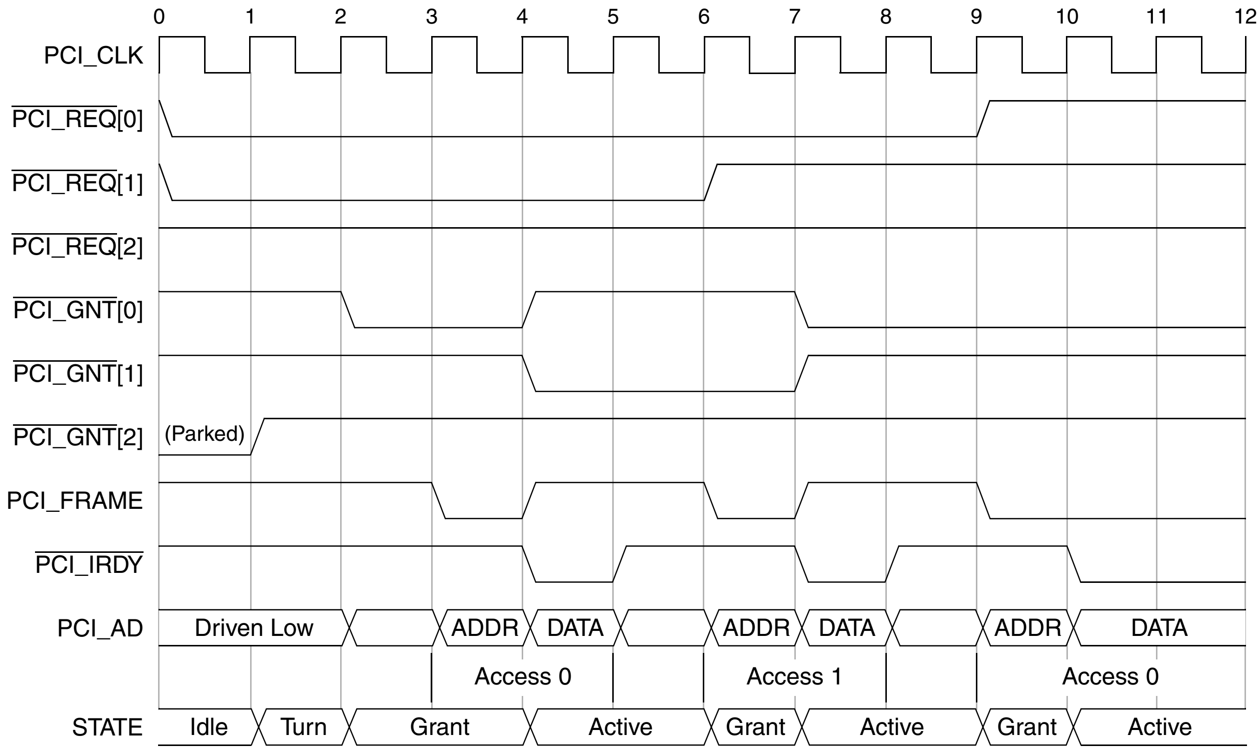


Figure 22-42. Alternating Priority

Device 0 and device 1 assert PCI_REQ while the bus parks with device 2. Because the PCI bus is idle, the arbiter deasserts PCI_GNT to the parked master (device 2) and a cycle later, it grants access to device 0. Device 0's transaction begins when PCI_FRAME asserts on clock 4. (The earliest device 0 can initiate a transaction on the PCI bus is the cycle following PCI_GNT assertion.) It leaves its PCI_REQ asserted to indicate it wants to perform another transaction. When PCI_FRAME asserts (PCI bus is active), hidden arbitration occurs and PCI_GNT to device 0 deasserts on the same cycle the arbiter asserts PCI_GNT to device 1. (Device 1 has priority because, of the two requesting masters [device 0 and device 1] device 1 is the least recently used.)

Device 0 completes its transaction on clock 5 and relinquishes the PCI bus. On clock 6, device 1 detects the PCI bus is idle (PCI_FRAME and PCI_IRDY deasserted) and because its PCI_GNT remains asserted, initiates the next transaction in the next cycle. To indicate it only requires this single transaction on the PCI bus, device 1 deasserts PCI_REQ on the same cycle it asserts PCI_FRAME.

Because device 0 is the only other requesting device, the arbiter asserts its PCI_GNT and leaves its PCI_GNT asserted until another request is detected.

Figure 22-43 starts out like Figure 22-42 with the bus parked with device 2 while device 0 and device 1 request use of the PCI bus (Assume device 0, device 1, and device 2 are assigned the same priority group and no other masters request use of the bus).

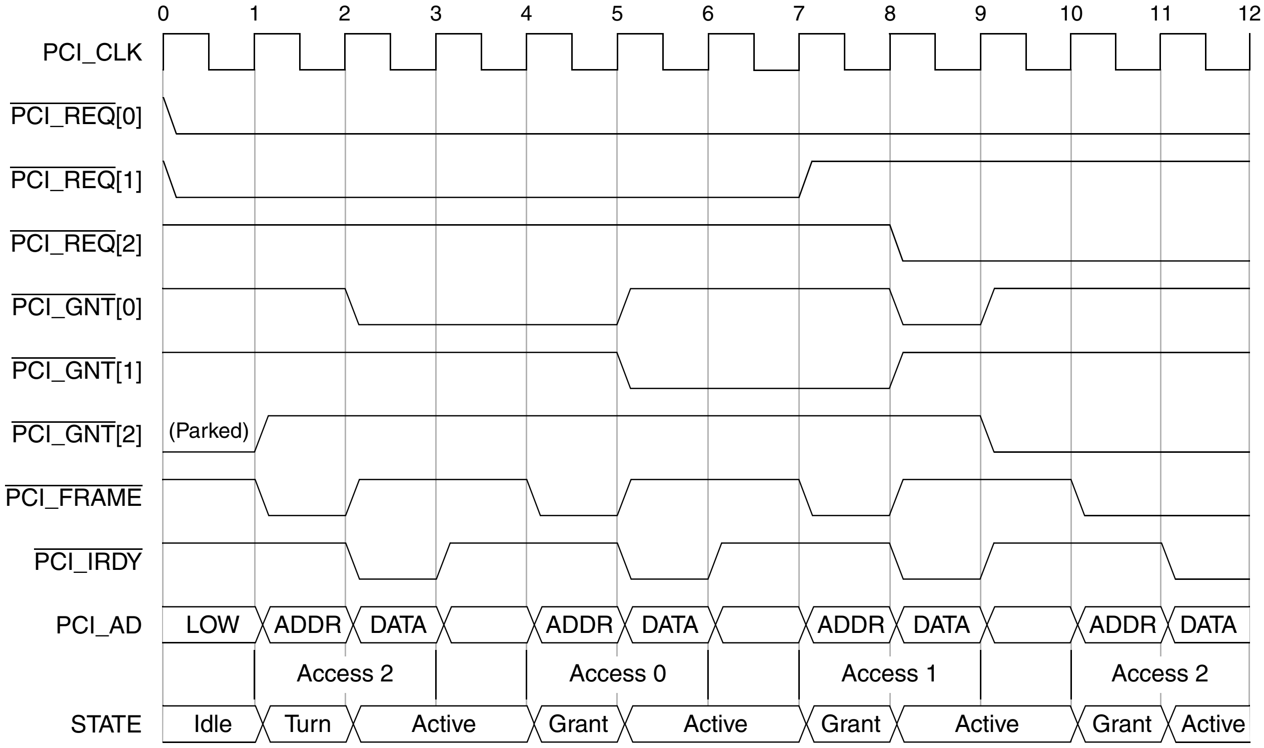


Figure 22-43. Higher Priority Override

The arbiter again deasserts device 2's PCI_GNT on clock 2, but device 2 initiates a transaction in the same cycle. As long as the PCI bus is idle and PCI_GNT is asserted, a master can begin a transaction on the next cycle. (Assertion of PCI_REQ is not required.)

Next access is again awarded to device 0 and upon detection of an idle PCI state, it performs its transaction (clocks 5 and 6). Because it has subsequent transactions to perform, device 0 leaves its PCI_REQ asserted. Like the previous timing diagram, PCI bus ownership switches to device 1.

While device 1 performs a transaction on the PCI bus on clock 8, device 0 is the only device requesting subsequent use of the bus. In the next cycle, the arbiter asserts PCI_GNT to device 0 in response to the request. During that same cycle, device 2 asserts its PCI_REQ. Because access 1 remains in progress, the arbiter determines device 2 is higher priority than device 0 (after device 1 access), rearbitrates and deasserts PCI_GNT to device 0, and asserts PCI_GNT to device 2 in the next cycle (clock 10).

22.4.5.6 Bus Parking

Bus parking asserts PCI_GNT to a PCI master while the PCI bus is idle and there are no incoming requests. While the bus parks on a master, the master must drive the PCI_AD, PCI_CBE, and PCI_PAR signals to prevent these signals from floating.

The PCI arbiter can be configured to park with the last master to use the bus or park with the PCI controller. The parking mode bit in the PCI arbiter control register, PACR[PKMD], determines the parking operation of the arbiter as shown in Table 22-34. In general, parking with the last master that acquired the bus can gain better system performance.

Table 22-34. PCI Arbiter Parking Mode Control

PACR[PKMD]	Parking Mode
0	Park with the last master to use the bus
1	Park with the internal PCI controller

22.4.5.6.1 Park With Last Master

The PCI arbiter continues issuing PCI_GNT to the previous master in the absence of any requests from other masters. There is no latency due to arbitration if the next master to request ownership of the bus is the master that last used the bus.

22.4.5.6.2 Park With The Internal PCI Controller

The arbiter monitors for PCI bus idle state (PCI_FRAME and PCI_IRDY deasserted) before parking the bus. The advantage to this method (versus not monitoring, where the arbiter asserts PCI_GNT to the PCI controller to park while the current master's transaction is active) is the current master is not forced off the bus due to master latency time-out. For example, if a single external master requests the bus while the PCI bus is idle, the arbiter deasserts PCI_GNT to the PCI controller and asserts PCI_GNT to the external master a cycle later. If the master does not have another transaction to run after the first one completes, it deasserts its PCI_REQ when it asserts PCI_FRAME.

While the transaction is in progress, the arbiter recognizes that the bus is busy and does not deassert the current master's PCI_GNT to park the bus with the PCI controller until the PCI bus is in an idle state (PCI_FRAME and PCI_IRDY deasserted). The current master can continue its transaction until it completes or until a request is received from another master. If the PCI arbiter removes PCI_GNT from the current master and issues PCI_GNT to the PCI controller (in absence of any requests from other masters) during the transaction, the current master is forced to relinquish the bus if its master latency timer exhausts before all data transfers complete. It would then have to wait two clocks and re-arbitrate for the bus again to resume the transaction at the point where it left off.

When the PCI controller comes out of reset, the internal PCI arbiter is set to park with the last master. To prevent the bus from floating before the bus activity occurs, the arbiter automatically parks the PCI bus ownership with the internal PCI controller. Therefore, the PCI controller drives the PCI_AD[31:0], PCI_CBE[3:0], and PCI_PAR signals after chip reset and before the first PCI bus request.

22.4.5.7 Master Time-Out

A master is broken if it has not initiated an access (dropped PCI_FRAME) after its PCI_GNT asserts (its PCI_REQ is also asserted) and the bus is in the idle state for 16 clocks. A 16 clock (PCI clock) timer institutes to prevent arbitration lock-up for this case. When the timer expires, the arbiter removes the PCI_GNT from the device and gives the bus to the master with the next highest priority. Subsequent requests from the timed-out master are ignored until its PCI_REQ is negated for at least one clock cycle.

A status bit is set when any master times out. If the corresponding interrupt enable bit is set, a CPU interrupt asserts. Software can query the status bits to detect a broken master in the PCI system. (See Section 22.3.3.2, “PCI Arbiter Status Register (PASR).”)

If a master does not initiate a transaction after its $\overline{\text{PCI\_GNT}}$ aasserts, but deasserts $\overline{\text{PCI\_REQ}}$ before the 16 clock timer expires, the arbiter deasserts $\overline{\text{PCI\_GNT}}$ and re-arbitrates for the next transaction. The master is not broken, and subsequent requests are acknowledged. This never-mind scenario is detrimental to system performance, however, and not a recommended implementation.

When PACR[RA] is set, the reset arbiter bit in the PCI arbiter control register resets the arbitration logic, ignore conditions, and timer. All status bits and corresponding interrupts are cleared. When the PACR[RA] bit subsequently clears to enable arbitration, requests from any previously detected broken master once again are recognized.

22.4.6 PCI Clock Scheme

The PCI controller requires a clock generated by an external source to input to the PCI_CLK signal to generate an internal PCI clock. The device uses this clock as its VCO reference clock. The internal PLL generates the internal PCI clock and all other clocks for the system. The PCIGSCR register reflects the PLL programmed ratios.

Table 22-35. PCI and System Clock Frequencies[1]

PCI_CLK	InternalBus CLK	Core Clock	Internal Bus Multiplier	PCIGSCR[AUTODIV] Internal to PCICLK differential
33 MHz	66 MHz	133 MHz	2	0x2
66 MHz	66 MHz	133 MHz	1	0x1
20 MHz	100 MHz	200 MHz	5	0x5
25 MHz	100 MHz	200 MHz	4	0x4
50 MHz	100 MHz	200 MHz	2	0x2
33 MHz	100 MHz	200 MHz	3	0x3
66 MHz	100 MHz	200 MHz	3/2	0x6
26 MHz	133 MHz	266 MHz	5	0x5
33 MHz	133 MHz	266 MHz	4	0x4
44 MHz	133 MHz	266 MHz	3	0x3
66 MHz	133 MHz	266 MHz	2	0x2

[1] 66 MHz operation is subject to system clocking restraints.

The PCI bus clock to external PCI devices generates from an external PLL, while the internal PCI clock generates from the Coldfire processor’s internal PLL. The internal bus clock is always faster than the PCI clock.

22.4.7 Interrupts

22.4.7.1 PCI Bus Interrupts

The PCI controller can generate an interrupt on the PCI bus interrupt signal, PCI_INTA. If using PCI_INTA, the PCI_INTA function must be selected for the pin in the port control logic for this device (see Chapter 16, “Pin Multiplexing and Control”).

22.4.7.2 Internal Interrupt

The PCI module can generate an interrupt to device’s interrupt controller. This interrupt is a logical OR of all of the available interrupt conditions of the PCI controller: detected PCI_PERR or PCI_SERR assertion, discarded target delayed reads, target error responses, internal bus initiated retry errors, target aborts, and initiator (master) aborts. See Section 22.3.2.1, “Global Status/Control Register (PCIGSCR),” Section 22.3.2.7, “Initiator Control Register (PCIICR),” and Section 22.3.2.8, “Initiator Status Register (PCIISR),” for more information. For the internal PCI arbiter, an internal interrupt can be enabled for detected master time-out conditions. For more information, see the enable and status registers for the PCI arbiter.

When status bits generate internal interrupts because the corresponding interrupt enable bit was set, clearing the status bit can deassert the interrupt output. Another way to force interrupt to a level low is to disable the interrupt enable that corresponds to the asserted status bit. The status bit, however, remains set.

22.4.8 Reset

Processor’s system reset provides reset capability. This signal resets hardware and software registers in the internal PCI arbiter.

A bit in the register space, PCIGSCR[PR], controls PCI_RST. During the system reset, this bit is set and PCI_RST asserts. No PCI traffic can occur during this time. Access to all internal registers is supported regardless of the value held in PCIGSCR[PR]. Only an internal software write of zero brings the PCI bus out of reset. See Section 22.3.2.1, “Global Status/Control Register (PCIGSCR).”

When PCIGSCR[PR] is clear and a 1 is written to the bit, all PCI Type 0 configuration registers load their reset values.

Because the external PCI_GNT[3:0] signals must tristate during PCI reset, the PCI_RST output signal is used as an output enable (active high) for all PCI_GNT[3:0] outputs.

22.5 Application Information

This section provides example usage of some features of the PCI module.

22.5.1 Internal Bus-Initiated Transaction Mapping

The use of the PCI configuration address register along with the initiator window registers provides many possibilities for PCI command and address generation. Table 22-16 shows how the PCI controller accepts read and write requests from an internal bus master and decodes them to different address ranges resulting in the generation of memory, I/O, configuration, interrupt acknowledge and special cycles on the PCI bus. The window registers are defined in Section 22.3.2.5, “Initiator Window n Base/Translation Address Register (PCIIWnBTAR),” and Section 22.3.2.6, “Initiator Window Configuration Register (PCIIWCR).”

Table 22-36. Transaction Mapping: Internal bus to PCI

Internal Bus Transaction (Internal Bus Slave Interface)	Cache Line Size, PCICR1[CLS] = 4	Initiator Register Settings				PCI Transaction Controller (Internal Bus Initiator Interface) → PCI Target
		Initiator Window Configuration, PCIIWCR		Configuration Address Register, PCICAR		
		IO/M#	PRC	E	Device Number = 1_1111	
Single-Beat 1 → 4 byte Read	x	0	00	x	x	MEMORY READ
Burst Read (16 bytes)	x	0	00	x	x	MEMORY READ
Single-Beat 1 → 4 byte Read	x	0	01	x	x	MEMORY READ
Burst Read	False	0	01	x	x	MEMORY READ
Burst Read	True	0	01	x	x	MEMORY READ LINE
Single-Beat 1 → 4 byte Read	x	0	10	x	x	MEMORY READ MULTIPLE
Burst Read	x	0	10	x	x	MEMORY READ MULTIPLE
Single-Beat 1 → 4 byte, or Burst Write	x	0	x	x	x	MEMORY WRITE
Single-Beat 1 → 4 byte Read	x	1	x	0	x	I/O READ
Single-Beat 1 → 4 byte Write	x	1	x	0	x	I/O WRITE
Single-Beat 1 → 4 byte Read	x	1	x	1	False	CONFIGURATION READ
Single-Beat 1 → 4 byte Write	x	1	x	1	False	CONFIGURATION WRITE
Single-Beat 1 → 4 byte Read	x	1	x	1	True	INTERRUPT ACKNOWLEDGE
Single-Beat 1 → 4 byte Write	x	1	x	1	True	SPECIAL CYCLE

- dual address cycles and memory write and invalidate commands are not supported
- x means don’t care

22.5.2 Address Translation

22.5.2.1 Inbound Address Translation

The PCI controller-as-target occupies up to as many as six memory target address windows on the PCI bus. Values programmed to the PCIBAR*n* registers of the PCI Type 00h configuration space determine location. These inbound memory window sizes are fixed in size. The PCISCR[M] bit must be set before any inbound memory window can be accessed. Otherwise, the PCI controller does not respond to PCI memory transactions.

PCI inbound address translation allows address translation to any space in local memory (4 Gbytes of address space). The target base address translation registers, PCITBATR*n*, specify the location of the inbound memory window. These registers are described in Section 22.4.2, "Configuration Interface." Address translation occurs for all enabled inbound transactions. If the PCITBATR*n*[EN] bit is cleared, the PCI controller aborts all PCI memory transactions to that base address window.

NOTE

> The PCI configuring master can program the PCIBAR*n* registers to overlapping PCI addresses. The default address translation value is PCITBATR5 in that case. It is not recommended to program overlapping PCIBAR space or overlapping PCITBATRs. An overlap of the PCITBATR*n* registers can cause data write-over of lower PCIBAR data.

The initiator window base address registers, PCIIW*n*BTAR, decode internal bus addresses for PCI bus transactions. The base address and base address mask values define the upper byte of address to decode. The internal bus address space dedicated to PCI transactions can be mapped to three 16-Mbyte or larger address spaces in the device. Initiator windows can be programmed to overlap, though not recommended. Priority for the windows is 0, 1, 2. Initiator window 0 has priority over all others and window 1 has priority over window 2.

In normal operation, software must not program either target address window translation register to address initiator window space. In that event, a PCI controller-as-target transaction propagates through the device's internal bus and requests PCI bus access as the PCI initiator. The PCI arbiter could see the PCI bus as busy (target read transaction in progress) and only a time-out frees the PCI bus.

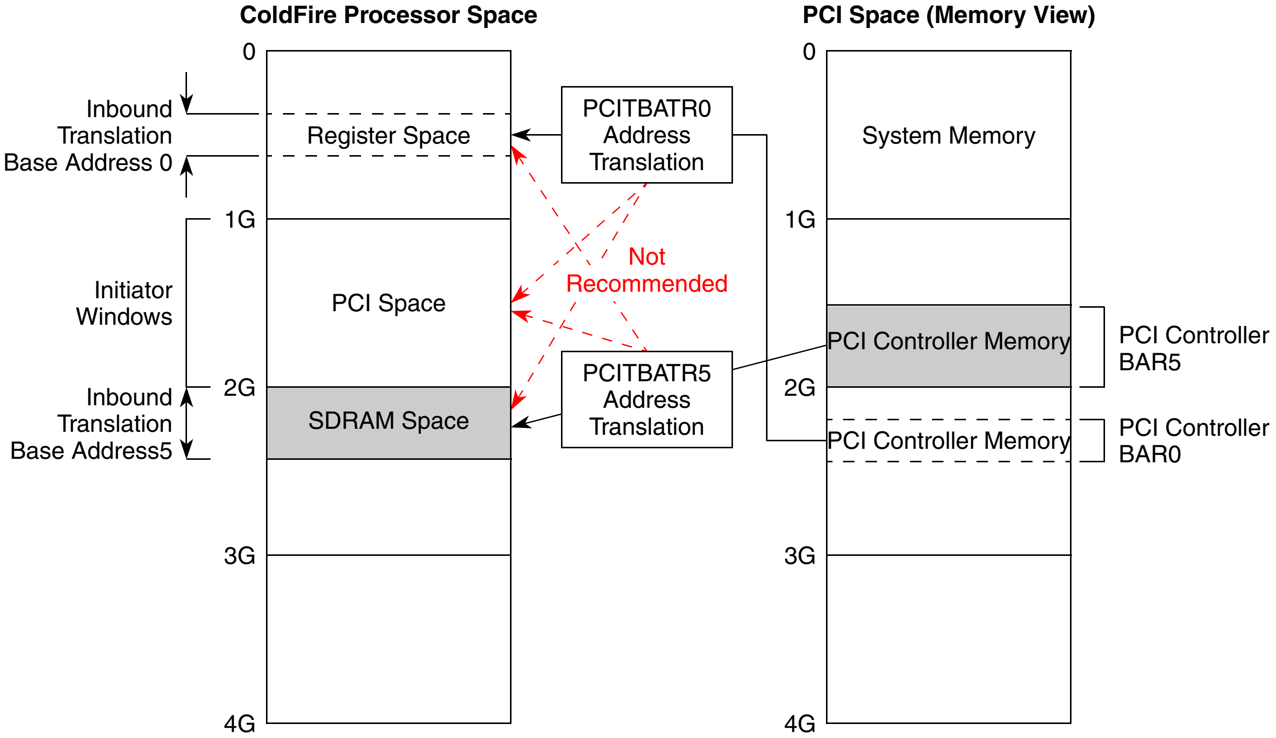


Figure 22-44. Inbound Address Map

22.5.2.2 Outbound Address Translation

Figure 22-45 shows an example of internal bus initiator window configurations. Overlapping the inbound memory window (local memory) and the outbound translation window is not supported and can cause unpredictable behavior. Figure 22-45 does not show the configuration mechanism.

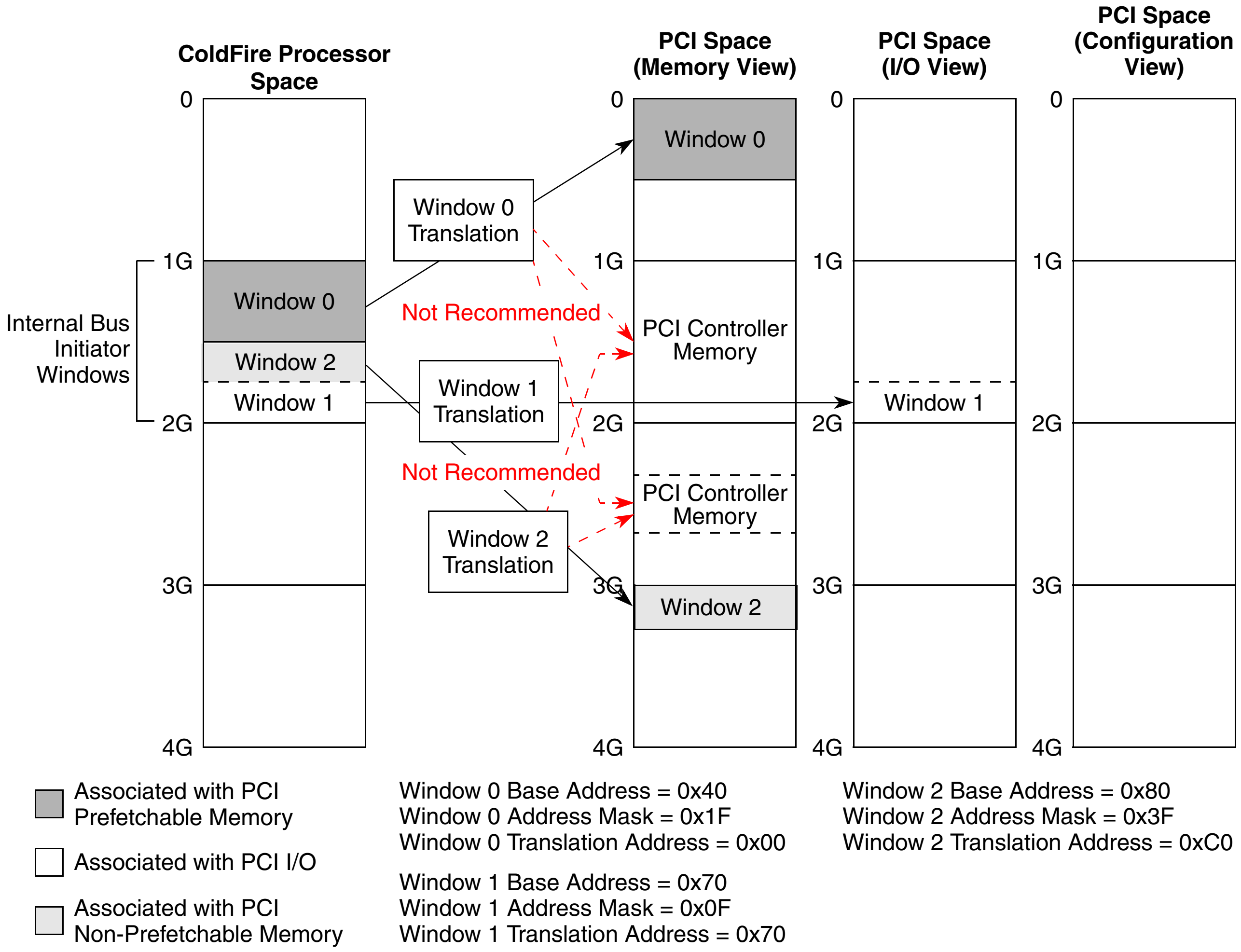

ColdFire Processor Space
PCI Space (Memory View)
PCI Space (I/O View)
PCI Space (Configuration View)
0
1G
2G
3G
4G
Internal Bus Initiator Windows
Window 0
Window 2
Window 1
Window 0 Translation
Window 1 Translation
Window 2 Translation
Not Recommended
Not Recommended
PCI Controller Memory
PCI Controller Memory
Window 0
Window 1
Window 2
Associated with PCI Prefetchable Memory
Associated with PCI I/O
Associated with PCI Non-Prefetchable Memory
Window 0 Base Address = 0x40
Window 0 Address Mask = 0x1F
Window 0 Translation Address = 0x00
Window 1 Base Address = 0x70
Window 1 Address Mask = 0x0F
Window 1 Translation Address = 0x70
Window 2 Base Address = 0x80
Window 2 Address Mask = 0x3F
Window 2 Translation Address = 0xC0

Figure 22-45. Outbound Address Map

22.5.2.3 Base Address Register Overview

Table 22-37 shows the available accessibility for all PCI associated base address and translation address registers in the PCI controller.

Table 22-37. Address Register Accessibility

Base Address Register	Register Function	PCI Bus Configuration Access	Processor Access	Any Internal Bus Master Access
PCIBAR*n*	PCI Base Address Register *n*	X	X	X
PCITBATR*n*	PCI Target Base Address Translation Register *n*		X	X
PCIIW*n*BTAR	Initiator Window Base/Translation Address Register *n*		X	X

22.5.2.4 Status Register Overview

Table 22-38 shows the associated bus interface(s) for each status register of the PCI controller. The table also names the interrupt flagged for the associated status register if enabled. PCIGSCR and PCIISR status bits can generate interrupts, but PCISCR bits cannot.

Table 22-38. Status Register Summary

Status Register	Register Function	Internal Bus Target Interface	Internal Bus Initiator Interface	Interrupt Generated
PCISCR	PCI Status/Command Register	X	X	None
PCIGSCR	PCI Global Status/Control Register	X	X	PCI controller interrupt
PCIISR	PCI Initiator Status Register		X	PCI controller interrupt

Chapter 23
Advanced Technology Attachment (ATA)

23.1 Introduction

This chapter discusses the modes of operation, signal descriptions, programming model, timing parameters, and functional description of the ATA interface. Figure 23-1 illustrates the block diagram of the interface.

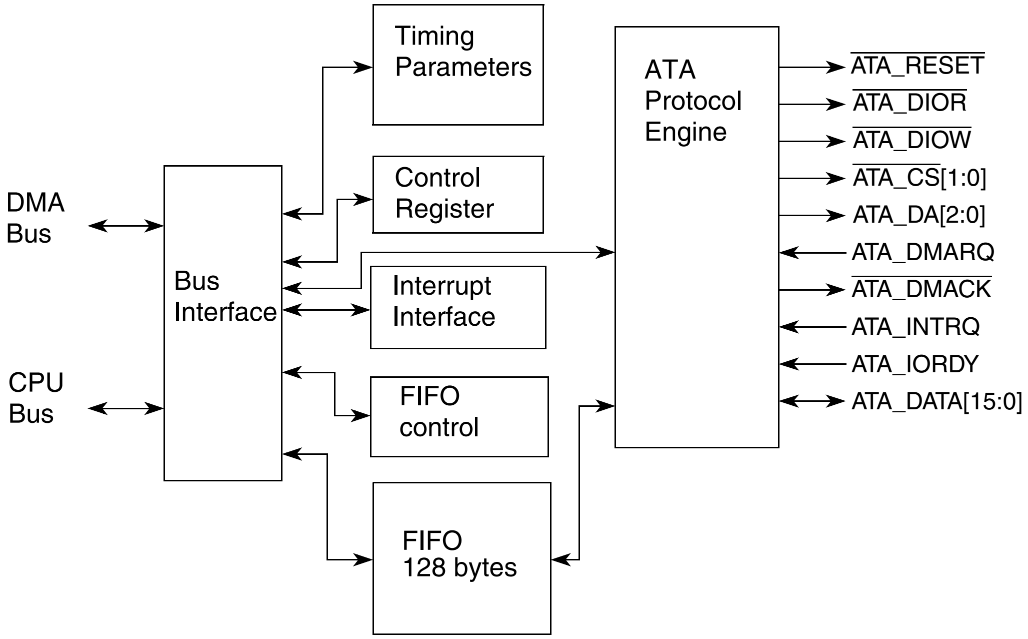


Figure 23-1. ATA Interface Block Diagram

23.1.1 Overview

The ATA block is an AT attachment host interface. It mainly interfaces with IDE hard disc drives and ATAPI optical disc drives. It interfaces with the ATA device over a number of ATA signals.

The ATA interface is compliant to the ATA-6 standard and supports the following protocols:

- PIO mode 0, 1, 2, 3 and 4
- Multiword DMA mode 0, 1 and 2
- Ultra DMA modes 0, 1, 2, 3 and 4 with bus clock of 50 MHz or higher
- Ultra DMA mode 5 with bus clock of 80 Mhz or higher

The ATA interface has 2 busses connected:

- One CPU bus for communication with the host processor
- One DMA bus for communication with the host DMA unit

All internal registers are visible from both busses, allowing smart DMA access to program the interface.

Before accessing the ATA bus, the host must program the timing parameters on the ATA bus. The timing parameters control the timing on the ATA bus. Most timing parameters are programmable as a number of clock cycles (1 to 255). Some are implied.

After programming the timing parameters, two protocols can be active at the same time on the ATA bus: PIO or DMA mode. All transfers between the FIFO and host or DMA busses are zero wait states transfer, so high speed transfer between FIFO and DMA/host bus is possible.

When a PIO access performed during a running DMA transfer, the DMA transfer pauses, the PIO access finishes, and the DMA transfer resumes again.

NOTE

The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, "Pin Multiplexing and Control") prior to configuring the ATA.

23.1.2 Features

The ATA interface has these features:

- Programmable timing on the ATA bus. Works with wide range of bus frequencies.
- Compliant with *AT Attachment - 6 with Packet Interface* specification found at http://www.t13.org
  - Supports programmed input/output (PIO) modes 0, 1, 2, 3, and 4
  - Supports multiword DMA modes 0, 1, and 2
  - Supports ultra DMA modes 0, 1, 2, 3, 4, and 5
- 128-byte FIFO interface
- FIFO receive alarm, FIFO transmit alarm, and FIFO end of transmission alarm to DMA unit
- Zero-wait cycles transfer between DMA bus and FIFO allows fast FIFO reading/writing

23.1.3 Modes of Operation

The interface offers two operating modes:

- PIO Mode

  An access to the ATA bus in PIO mode occurs when the host CPU or the host DMA unit reads or writes ATA PIO register. During PIO transfer the incoming device bus cycle translates to an ATA PIO bus cycle by the ATA protocol engine. No buffering of data occurs, so the host CPU or host DMA cycle stalls until the ATA bus read data is available on read or stalls until the device bus data can be put on the ATA bus during write.

  PIO accesses can happen to the bus at any time, even during a running ATA DMA transfer. In this case, the DMA transfer pauses, the PIO cycle completes, and the DMA transfer resumes.

The PIO mode is a slow protocol, mainly intended to program the ATA disc drive, but also possible to transfer data to/from the disc drive. During PIO mode, the FIFO is not active.

- DMA Mode

  In DMA mode, data transfers between the ATA bus and the FIFO. Two different DMA protocols are supported on the ATA bus: ultra DMA mode and multiword DMA mode. Acontrol register bit selects which DMA mode.

  A DMA transfer starts when DMA mode transfer is enabled by writing some control bit and when the drive connected to the ATA bus pulls its DMARQ signal high.

  During an ATA bus DMA transfer, data transfers between the ATA bus and the FIFO. The transfer pauses to avoid FIFO overflow and FIFO underflow.

  The host CPU or the host smart DMA unit must read data or write data to the FIFO to keep the transfer going. Normally, the host (smart) DMA unit takes on this task. For this purpose, the FIFO receive/transmit alarms are sent to the host DMA unit. The FIFO receive alarm informs the host DMA unit of at least one packet of data waiting in the FIFO to be read by the host DMA. When this alarm is asserted, the host DMA should transfer one packet of data from FIFO to the main memory. Typical packet size is 32 bytes (8 longwords), but other packet sizes can be managed too. FIFO transmit alarm informs the host DMA unit of space for at least one packet to be written by the host DMA. When this alarm is asserted, the host DMA must transfer one packet of data from main memory to the FIFO. Typical packet size is 32 bytes (8 longwords), but other packet sizes can be managed too.

23.2 External Signal Description

See Table 23-1 for the list of signals entering and exiting this module to peripherals within the device.

Table 23-1. ATA Signal Properties

Name	Function	Reset State	I/O
$\overline{\text{ATA\_RESET}}$	ATA bus reset signal[1]	0	O
$\overline{\text{ATA\_DIOR}}$	ATA bus read strobe	1	O
$\overline{\text{ATA\_DIOW}}$	ATA bus write strobe	1	O
$\overline{\text{ATA\_CS}}$[1:0]	ATA bus chip selects	1	O
ATA_DA[2:0]	ATA bus address line	0	O
ATA_DMARQ	ATA bus DMA request	—	I/O
$\overline{\text{ATA\_DMACK}}$	ATA bus DMA acknowledge	1	O
ATA_INTRQ	ATA bus interrupt request	—	I/O
ATA_IORDY	ATA bus I/O channel ready	—	O
ATA_DATA[15:0]	ATA data bus (little-endian)	Hi-Z	Tri-state I/O

[1] This signal is a standard ATA bus signal. It conforms with the ATA specification.

23.2.1 Detailed Signal Descriptions

For a detailed description of the ATA bus signal, refer to the ATA-6 specification.

23.2.1.1 ATA Reset ($\overline{\text{ATA\_RESET}}$)

This output signal is the ATA reset signal. When asserted, the ATA bus is in reset state. When negated, no reset. The ATA bus is in reset when the appropriate bit in the control register is cleared. After system reset, the ATA bus is in reset.

23.2.1.2 ATA DIO Read ($\overline{\text{ATA\_DIOR}}$)

This output signal corresponds to the ATA signal DIOR. During PIO and multiword DMA transfer, function is read strobe. During ultra DMA in burst, function is HDMARDY. During ultra DMA out burst, function is host strobe.

23.2.1.3 ATA DIO Write ($\overline{\text{ATA\_DIOW}}$)

This output signal corresponds to ATA signal DIOW. During PIO and multiword DMA transfer, function is write strobe. During ultra DMA burst, function is STOP, signalling when host wants to terminate running ultra DMA transfer.

23.2.1.4 ATA Chip Selects ($\overline{\text{ATA\_CS}}$[1:0])

These output signals are the chip selects for the ATA bus.

23.2.1.5 ATA Address Lines (ATA_DA[2:0])

These output signals are the address lines of the ATA bus.

23.2.1.6 ATA DMA Request (ATA_DMARQ)

This input signal is the ATA bus device DMA request. The device asserts it if it wants to transfer data using multiword DMA or ultra DMA mode

23.2.1.7 ATA DMA Acknowledge ($\overline{\text{ATA\_DMACK}}$)

This output signal is the ATA bus host DMA acknowledge. The host negates it when it grants the DMA request.

23.2.1.8 ATA Interrupt Request (ATA_INTRQ)

This input signal is the ATA bus interrupt request. The device asserts it when it wants to interrupt the host CPU.

23.2.1.9 ATA I/O Ready (ATA_IORDY)

This input signal is the ATA bus I/O channel ready signal. It has three functions:

- IORDY — active low wait during PIO cycles
- DDMARDY — active low device ready during ultra DMA out transfers
- DSTROBE — device strobe during ultra DMA in transfers

23.2.1.10 ATA Data Bus (ATA_DATA[15:0])

This is the bidirectional, tri-state ATA data bus

23.3 Memory Map/Register Definition

The following section provides the detailed descriptions for all of the ATA registers. Table 23-2 shows the ATA memory map.

Table 23-2. ATA Memory Map

Address	Register	Width	Access	Reset Value	Section/Page
Timing Registers					
0x9000_0000	TIME_OFF—t_{off} transceiver timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0001	TIME_ON—t_{on} transceiver timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0002	TIME_1—t_1 PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0003	TIME_2W—t_2 write PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0004	TIME_2R—t_2 read PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0005	TIME_AX—t_A PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0006	TIME_PIORDX—t_{rd} PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0007	TIME_4—t_4 PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0008	TIME_9—t_9 PIO timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0009	TIME_M—t_m MDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_00A	TIME_JN—t_n and t_j MDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_000B	TIME_D—t_d MDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_000C	TIME_K—t_k MDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_000D	TIME_ACK—t_{ack} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_000E	TIME_ENV—t_{env} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_000F	TIME_RPX—t_{rp} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0010	TIME_ZAH—t_{zah} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0011	TIME_MLIX—t_{mli} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0012	TIME_DVH—t_{dvh} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0013	TIME_DZFS—t_{dzfs} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0014	TIME_DVS—t_{dvs} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0015	TIME_CVH—t_{cvh} UDMA timing register	8	R/W	0x01	23.3.2/23-7

Table 23-2. ATA Memory Map (continued)

Address	Register	Width	Access	Reset Value	Section/Page
0x9000_0016	TIME_SS—t_{ss} UDMA timing register	8	R/W	0x01	23.3.2/23-7
0x9000_0017	TIME_CYC—t_{cyc} and t_{2cyc} UDMA timing register	8	R/W	0x01	23.3.2/23-7
FIFO Registers					
0x9000_0018	FIFO_DATA32—32-bit wide data port to/from FIFO	32	R/W	Undefined	23.3.3/23-7
0x9000_001C	FIFO_DATA16—16-bit wide data port to/from FIFO	16	R/W	Undefined	23.3.3/23-7
0x9000_0020	FIFO_FILL—FIFO filling in halfwords	8	R	0x00	23.3.4/23-8
ATA Interface Registers					
0x9000_0024	ATA_CR—ATA interface control register	8	R/W	0x00	23.3.5/23-8
0x9000_0028	ATA_ISR—Interrupt status register	8	R	See Section	23.3.6.1/23-10
0x9000_002C	ATA_IER—Interrupt enable register	8	R/W	See Section	23.3.6.2/23-11
0x9000_0030	ATA_ICR—Interrupt clear register	8	W	Undefined	23.3.6.3/23-12
0x9000_0034	FIFO_ALARM—FIFO alarm threshold	8	R/W	0x00	23.3.7/23-12
Drive Registers					
0x9000_00A0	DRIVE_DATA—Drive data register	16	R/W	See Spec[1]	23.3.8/23-12
0x9000_00A4	DRIVE_FEATURES—Drive features register	8	R/W	See Spec[1]	23.3.8/23-12
0x9000_00A8	DRIVE_SECTOR_COUNT—Drive sector count register	8	R/W	See Spec[1]	23.3.8/23-12
0x9000_00AC	DRIVE_LBA_LOW—Drive LBA low register	8	R/W	See Spec[1]	23.3.8/23-12
0x9000_00B0	DRIVE_LBA_MID—Drive LBA middle register	8	R/W	See Spec[1]	23.3.8/23-12
0x9000_00B4	DRIVE_LBA_HIGH—Drive LBA high register	8	R/W	See Spec[1]	23.3.8/23-12
0x9000_00B8	DRIVE_DEV_HEAD—Drive device head register	8	R/W	See Spec[1]	23.3.8/23-12
0x9000_00BC	DRIVE_STATUS—Drive status register	8	R	See Spec[1]	23.3.8/23-12
	DRIVE_COMMAND—Drive command register	8	W	See Spec[1]	23.3.8/23-12
0x9000_00D8	DRIVE_ALT_STATUS—Drive alternate status register	8	R	See Spec[1]	23.3.8/23-12
	DRIVE_CONTROL—Drive control register	8	W	See Spec[1]	23.3.8/23-12

[1] See the *AT Attachment - 6 with Packet Interface* specification available at http://www.t13.org.

23.3.1 Endianness

The ATA interface works in big endian mode.The few 16-bit and 32-bit registers represent strings of 2 or 4 bytes. The byte order in the 16-bit or 32-bit register is as follows:

- Big endian, 32-bit register
  - — bits [31:24]: byte 0
  - — bits [23:16]: byte 1

  - bits [15:8]: byte 2
  - bits [7:0]: byte 3
- Big endian, 16-bit register
  - bits [15:8]: byte 0
  - bits [7:0]: byte 1

23.3.2 Timing Registers (TIME_*x*)

The timing registers contain the parameters that control the timing on the ATA bus. Details of the ATA bus timing are shown in detail in the following sections:

- Section 23.4.1.1, "PIO Mode Timing Diagrams"
- Section 23.4.1.2, "Multiword DMA Mode Timing Diagrams"
- Section 23.4.1.3, "Ultra DMA In Timing Diagrams"
- Section 23.4.1.4, "Ultra DMA Out Timing Diagrams"

Every timing parameter is 8-bit wide and can assume valid values between 1 and 255. Reset value is always 0x01.

Address: 0x9000_0000 (TIME_OFF)
0x9000_0001 (TIME_ON)
0x9000_0002 (TIME_1)
0x9000_0003 (TIME_2W)
0x9000_0004 (TIME_2R)
0x9000_0005 (TIME_AX)
0x9000_0006 (TIME_PIO_RDX)
0x9000_0007 (TIME_4)
0x9000_0008 (TIME_9)
0x9000_0009 (TIME_M)
0x9000_000A (TIME_JN)
0x9000_000B (TIME_D)
0x9000_000C (TIME_K)
0x9000_000D (TIME_ACK)
0x9000_000E (TIME_ENV)
0x9000_000F (TIME_RPX)
0x9000_0010 (TIME_ZAH)
0x9000_0011 (TIME_MLIX)
0x9000_0012 (TIME_DVH)
0x9000_0013 (TIME_DZFS)
0x9000_0014 (TIME_DVS)
0x9000_0015 (TIME_CVH)
0x9000_0016 (TIME_SS)
0x9000_0017 (TIME_CYC)

Access: User read/write

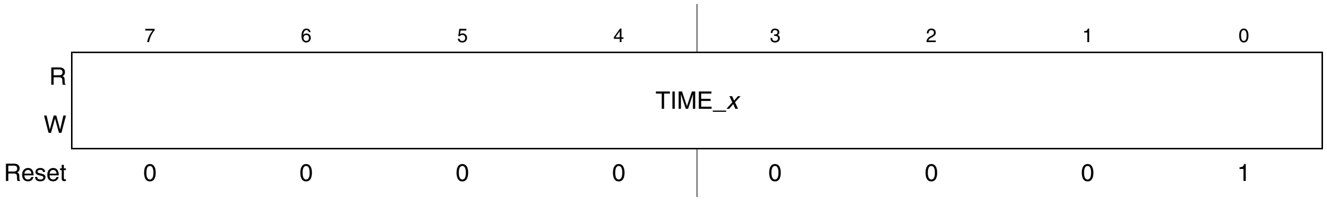


Figure 23-2. TIME_*x* Register

Table 23-3. TIME_*x* Field Descriptions

Field	Description
7–0 TIME_*x*	Indicates the timing parameter used for the ATA bus. See Section 23.4.1, "Timing on ATA Bus," for details on setting the individual values. Valid values between 1 and 255. Reset value is always 0x01.

23.3.3 FIFO Data Register (FIFO_DATA_*n*)

The FIFO data register reads or writes data to the internal FIFO. It can be accessed as a 16-bit register or as a 32-bit register. Any longword write to the register puts the four bytes written into the FIFO, and any word write puts the two bytes into the FIFO. Likewise, any longword read reads four bytes from the FIFO and any word read reads two bytes from the FIFO.

The following two figures show the FIFO data register in 16- and 32-bit modes.

Address: 0x9000_00018 (FIFO_DATA32) Access: User 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
R W	FIFO_DATA																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 23-3. FIFO Data Register in 32-Bit Mode

Address: 0x9000_0001C (FIFO_DATA16) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	FIFO_DATA															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 23-4. FIFO Data Register in 16-bit Mode

23.3.4 FIFO_FILL Register

See Figure 23-5 for illustration of valid bits in the FIFO_FILL register and Table 23-2 for description of the bit fields.

Address: 0x9000_0020 (FIFO_FILL) Access: User read-only

	7	6	5	4	3	2	1	0
R	FIFO_FILL							
W								
Reset	0	0	0	0	0	0	0	0

Figure 23-5. FIFO_FILL Register

FIFO_FILL is a read-only register. Any read to it returns the current number of halfwords present in the FIFO.

23.3.5 ATA Control Register (ATA_CR)

See Figure 23-6 for illustration of valid bits in the ATA control register and Table 23-4 for description of the bit fields.

Address: 0x9000_0024 (ATA_CR) Access: User read/write

	7	6	5	4	3	2	1	0
R W	FEN	RESET	FREFILL	FEMPTY	DMAPEND	DMAMODE	DMADIR	IORDYEN
Reset	0	0	0	0	0	0	0	0

Figure 23-6. ATA Control Register (ATA_CR)

Table 23-4. ATA_CR Field Descriptions

Field	Description
7 FEN	This field controls if the internal FIFO is in reset or enabled 0 FIFO reset 1 FIFO normal operation
6 RESET	Controls the level on the $\overline{\text{ATA\_RESET}}$ pin, and controls the reset of the internal ATA protocol engine. 0 $\overline{\text{ATA\_RESET}}$ = 0, ATA drive is reset, and internal protocol engine reset. 1 $\overline{\text{ATA\_RESET}}$ = 1, ATA drive is not reset, and internal protocol engine normal operation.
5 FREFILL	FIFO transmit enable. Controls if the FIFO makes transmit data requests to the DMA. If enabled, the FIFO requests the DMA to refill it when FIFO filling drops below the alarm level. 0 FIFO refill by DMA disabled 1 FIFO refill by DMA enabled
4 FEMPTY	FIFO receive enable. Controls if the FIFO makes receive data requests to the DMA. If enabled, the FIFO requests the DMA to empty it whenr FIFO filling becomes greater or equal to the alarm level. 0 FIFO empty by DMA disabled 1 FIFO empty by DMA enabled
3 DMAPEND	DMA pending bit. This bit controls if the ATA interface responds to a DMA request originating in the drive. If this bit asserts, the ATA interface starts a multiword DMA or ultra DMA burst when the drive asserts ATA_DMARQ. 0 ATA interface does not start DMA burst 1 ATA interface starts multiword DMA or ultra DMA burst when drive asserts ATA_DMARQ
2 DMAMODE	DMA mode. If a DMA burst starts, the UDMA or MDMA protocol is used. 0 Multiword DMA protocol is used 1 Ultra DMA protocol is used
1 DMADIR	DMA burst direction. Indicates the data direction on any DMA burst started. 0 DMA in burst, ATA interface reads from drive 1 DMA out burst, ATA interface writes to drive
0 IORDYEN	IORDY enable. Indicates if the ATA_IORDY handshake is used during PIO mode. 0 IORDY is disregarded 1 IORDY handshake is used

23.3.6 Interrupt Registers

A group of three registers controls the interrupt interface from the ATA module to the CPU's interrupt controller and DMA controller.

- Interrupt request. Bits 3–6 of the interrupt registers control ATA interrupts. A request to the interrupt controller generates if one of the four bits is set in the interrupt status register (ATA_ISR), while the same bit is set in the interrupt enable register (ATA_IER).

- DMA request. Bit 7 of the interrupt registers control ATA DMA request. If the DMA bit is set in the ATA_IER and ATA_ISR registers, a request is sent to the DMA controller. The goal of this request is to inform the DMA that running data transfer has ended.

These three interrupt registers have mostly the same bits. If a bit is set in the ATA_ISR register, its interrupt is pending and produces an interrupt if the corresponding bit is set in the ATA_IER register. Some bits in the ATA_ISR are sticky bits. Writing a 1 to the corresponding bit in the interrupt clear register (ATA_ICR) resets them.

23.3.6.1 Interrupt Status Register (ATA_ISR)

The interrupt status register reports the status of various conditions of the ATA controller and its FIFO.

Address: 0x9000_0028 (ATA_ISR)

Access: User read-only

	7	6	5	4	3	2	1	0
R	DMA	FUF	FOF	IDLE	INT			
W								
Reset	0[1]	0	0	1	0[1]	—	—	—

[1] Bits DMA and INT only reset to 0 if during reset the interrupt input is low.

Figure 23-7. Interrupt Status Register (ATA_ISR)

Table 23-5. ATA_ISR Field Descriptions

Field	Description
7 DMA	ATA DMA request. Reflects the value of the ATA_INTRQ interrupt input signal. When this bit is set in the ATA_ISR and ATA_IER registers, the DMA end-of-transfer request is sent. The interrupt clear register, ATA_ICR, has no influence on this bit. 0 ATA_INTRQ negated 1 ATA_INTRQ asserted
6 FUF	FIFO underfow. Sticky bit that reports FIFO underflow. It is cleared by writing a 1 to this bit in the ATA_ICR register. When this bit is set in the ATA_ISR and ATA_IER registers, an interrupt is requested to the CPU. 0 No FIFO underflow 1 FIFO underflow detected
5 FOF	FIFO overflow. Sticky bit that reports FIFO overflow. It is cleared by writing a 1 to this bit in the ATA_ICR register. When this bit is set in the ATA_ISR and ATA_IER registers, an interrupt is requested to the CPU. 0 No FIFO overflow 1 FIFO overflow detected
4 IDLE	Controller idle. Indicates the ATA protocol engine is idle (there is no activity on the ATA bus). When the bit is set in the ATA_ISR and ATA_IER registers, an interrupt is requested to the CPU. The ATA_ICR register has no influence on this bit. 0 Activity on the ATA bus 1 No activity on the ATA bus, engine is idle

Table 23-5. ATA_ISR Field Descriptions (continued)

Field	Description
3 INT	ATA interrupt request. Reflects the value of the ATA_INTRQ interrupt input. It is set when the drive interrupt is pending, cleared otherwise. It has the same functionality as the DMA bit, but affects the ATA interrupt to the CPU, while the other affects the ATA DMA request. When the bit is set in the ATA_ISR and ATA_IER registers, an interrupt is requested to the CPU, signalling the CPU the drive is requesting attention. The ATA_ICR register has no influence on this bit. 0 ATA_INTRQ negated 1 ATA_INTRQ asserted
2–0	Reserved.

23.3.6.2 Interrupt Enable Register (ATA_IER)

The ATA interrupt enable register enables requests to the interrupt and DMA controller.

Address: 0x9000_002C (ATA_IER) Access: User read/write

	7	6	5	4	3	2	1	0
R	DMA	FUF	FOF	IDLE	INT			
W								
Reset	0	0	0	0	0	—	—	—

Figure 23-8. ATA_IER Register

Table 23-6. Interrupt Enable Register Field Description

Field	Description
7 DMA	ATA DMA request enable. 0 ATA DMA request disabled 1 ATA DMA request enabled
6 FUF	FIFO underfow interrupt enable. 0 FIFO underflow interrupt disabled 1 FIFO underflow interrupt enabled
5 FOF	FIFO overflow interrupt enable. 0 FIFO overflow interrupt disabled 1 FIFO overflow interrupt enabled
4 IDLE	ATA controller idle interrupt enable. 0 Idle interrupt disabled 1 Idle interrupt enabled
3 INT	ATA interrupt request enable. 0 Interrupt request disabled 1 Interrupt request enabled
2–0	Reserved.

23.3.6.3 Interrupt Clear Register (ATA_ICR)

The interrupt clear register clears the FIFO underflow/overflow status bits in the ATA_ISR register.

Address: 0x9000_0030 (ATA_ICR) Access: User write-only

	7	6	5	4	3	2	1	0
R								
W		FUF	FOF					
Reset	—	—	—	—	—	—	—	—

Figure 23-9. ATA_ICR Register

Table 23-7. ATA_ICR Field Description

Field	Description
7	Reserved.
6 FUF	FIFO underfow clear. Writing a 1 to this bit clears the corresponding bit in the ATA_ISR. Writing a 0 has no effect.
5 FOF	FIFO overflow clear. Writing a 1 to this bit clears the corresponding bit in the ATA_ISR. Writing a 0 has no effect.
4–0	Reserved.

23.3.7 FIFO Alarm Register (FIFO_ALARM)

The FIFO alarm register contains threshold to generate the FIFO transmit and receive requests to the DMA controller:

- If (ATA_CR[FREFILL] == 1 and and FIFO_FILL < FIFO_ALARM) then request is sent to the DMA controller to refill the FIFO.
- If (ATA_CR[FEMPTY] == 1 and and FIFO_FILL ≥ FIFO_ALARM) then request is sent to the DMA controller to empty the FIFO.

Address: 0x9000_0034 (FIFO_ALARM) Access: User read/write

	7	6	5	4	3	2	1	0
R W	FIFO_ALARM							
Reset	0	0	0	0	0	0	0	0

Figure 23-10. FIFO_Alarm Register

23.3.8 Drive Registers

Some drive registers are addressable, but are not present in ATA interface module. Table 23-2 provides a list of these registers. If a read or write access is made to one of these registers, read or write maps to a PIO read or write cycle on the ATA bus, and the corresponding register in the device attached to the ATA bus is accessed.No description of operation of these registers is given here. Consult the ATA specification for details.

The module operates in big endian mode. Therefore, as the drive data register (ATA_DDR) bytes are transferred to/from the ATA bus, they switch order. No swaps occur for any other register.

23.4 Functional Description

The ATA interface provides two ways to communicate with ATA peripherals connected to ATA bus:

- PIO mode read/write operation to the ATA bus.
- DMA transfers with the ATA bus

The operation of the peripheral is described in detail in the following sections.

23.4.1 Timing on ATA Bus

Timing on the ATA bus is explained in this section with timing diagrams and equations.

23.4.1.1 PIO Mode Timing Diagrams

A timing diagram for PIO read mode appears in Figure 23-11.

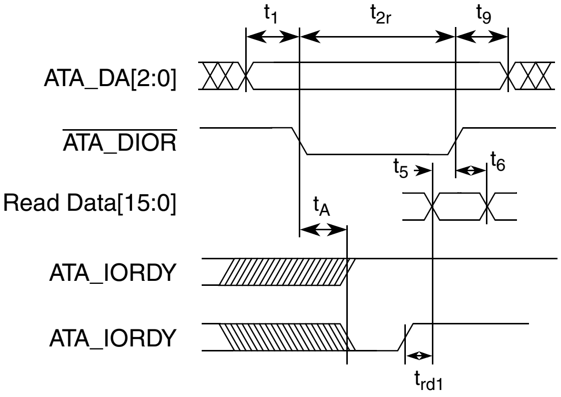


Figure 23-11. PIO Read Mode Timing

To fulfill read mode timing, the different timing parameters appearing in Table 23-8 must be observed.

Table 23-8. Timing Parameters PIO Read

ATA Parameter	PIO Read Mode Timing Parameter[1]	Value	How to Meet?
t1	t1	t1(min) = time_1 * T - (tskew1 + tskew2 + tskew5)	TIME_1
t2	t2r	t2(min) = time_2r * T - (tskew1 + tskew2 + tskew5)	TIME_2R
t9	t9	t9(min) = time_9 * T - (tskew1 + tskew2 + tskew6)	TIME_9
t5	t5	t5(min) = tco + tsu + tbuf + tbuf + tcable1 + tcable2	if not met, increase TIME_2

Table 23-8. Timing Parameters PIO Read (continued)

ATA Parameter	PIO Read Mode Timing Parameter[1]	Value	How to Meet?
t6	t6	0	
tA	tA	tA(min) = (1.5 + time_ax) * T - (tco + tsui + tcable2 + tcable2 + 2*tbuf)	TIME_AX

trd	trd1	trd1(max) = (-trd) + (tskew3 + tskew4)
		trd1(min) = (time_pio_rdx - 0.5)*T - (tsu + thi)
		(time_pio_rdx - 0.5) * T > tsu + thi + tskew3 + tskew4
			TIME_PIO_RDX

t0	-	t0(min) = (time_1 + time_2 + time_9) * T	TIME_1, TIME_2R, TIME_9

[1] See Figure 23-11.

In PIO write mode, timing waveforms are somewhat different. A timing diagram appears in Figure 23-12.

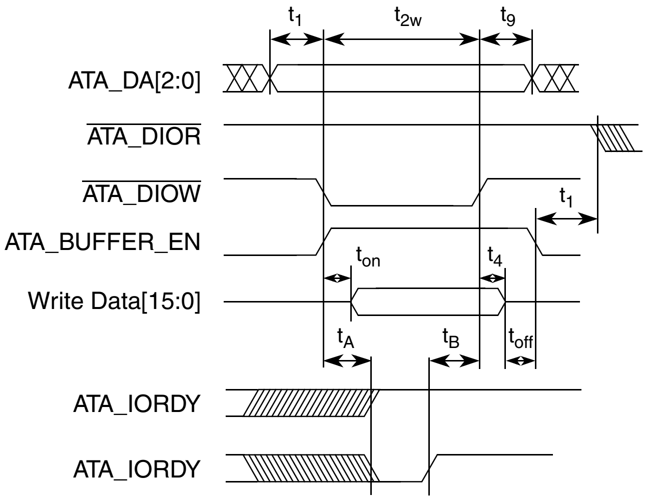


Figure 23-12. PIO Write Mode Timing

To fulfill this timing, several parameters need to be observed. See Table 23-9 for details on PIO write mode timing.

Table 23-9. Timing Parameters PIO Write

ATA Parameter	PIO Write Mode Timing Parameter[1]	Value	How to meet?
t1	t1	t1(min) = time_1 * T - (tskew1 + tskew2 + tskew5)	TIME_1
t2	t2w	t2(min) = time_2w * T - (tskew1 + tskew2 + tskew5)	TIME_2W
t9	t9	t9(min) = time_9 * T - (tskew1 + tskew2 + tskew6)	TIME_9
t3	-	t3(min) = (time_2w - time_on)* T - (tskew1 + tskew2 +tskew5)	If not met, increase TIME_2W

Table 23-9. Timing Parameters PIO Write (continued)

ATA Parameter	PIO Write Mode Timing Parameter[1]	Value	How to meet?
t4	t4	t4(min) = time_4 * T - tskew1	TIME_4
tA	tA	tA = (1.5 + time_ax) * T - (tco + tsui + tcable2 + tcable2 + 2*tbuf)	TIME_AX
t0	—	t0(min) = (time_1 + time_2 + time_9) * T	TIME_1, TIME_2R, TIME_9
—	—	Avoid bus contention when switching buffer on by making ton long enough	
—	—	Avoid bus contention when switching buffer off by making toff long enough	

[1] See Figure 23-12.

23.4.1.2 Multiword DMA Mode Timing Diagrams

In multiword DMA mode, see Figure 23-13 for read timing diagram and Figure 23-14 for write timing diagram.

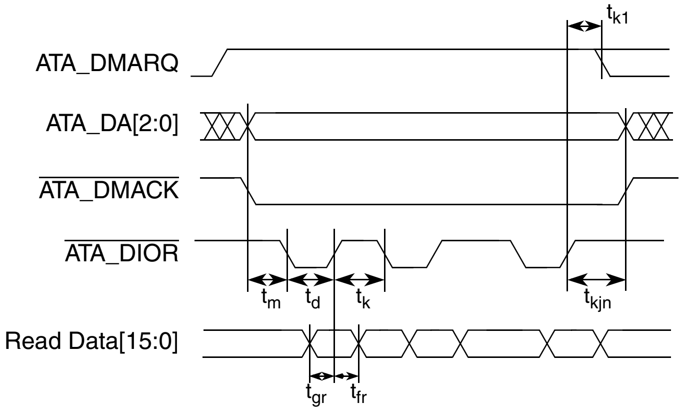


Figure 23-13. MDMA Read Timing

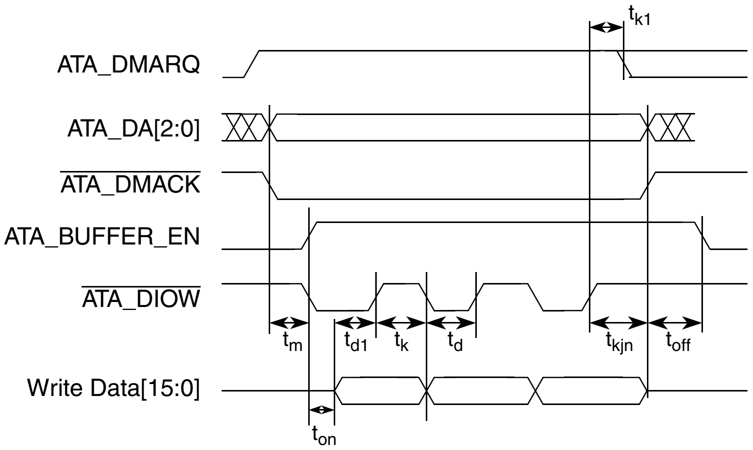


Figure 23-14. MDMA Write Timing

To meet the timing requirement, a number of timing parameters must be controlled. See Table 23-10 for details on timing parameters for MDMA read and write.

Table 23-10. Timing Parameters MDMA Read and Write

ATA Parameter	MDMA Read[1] and Write[2] TIming	Value	How to Meet?
tm, ti	tm	tm(min) = ti(min) = time_m * T - (tskew1 + tskew2 + tskew5)	TIME_M
td	td, td1	td1(min) = td(min) = time_d * T - (tskew1 + tskew2 + tskew6)	TIME_D
tk	tk	tk(min) = time_k * T - (tskew1 + tskew2 + tskew6)	TIME_K
t0	—	t0(min) = (time_d + time_k) * T	TIME_D, TIME_K
tg(read)	tgr	tgr(min-read) = tco + tsu + tbuf + tbuf + tcable1 + tcable2 tgr(min-drive) = td - te(drive)	TIME_D
tf(read)	tfr	tfr(min-drive) =0k	—
tg(write)	—	tg(min-write) = time_d * T -(tskew1 + tskew2 + tskew5)	TIME_D
tf(write)		tf(min-write) = time_k * T - (tskew1 + tskew2 + tskew6)	TIME_K
tL		tL(max) = (time_d + time_k-2)*T - (tsu + tco + 2*tbuf + 2*tcable2)	TIME_D, TIME_K[3]
tn, tj	tkjn	tn= tj= tkjn = (max(time_k,. time_jn) * T - (tskew1 + tskew2 + tskew6)	TIME_JN

	ton
	toff
		ton = time_on * T - tskew1
		toff = time_off * T - tskew1	

[1] See Figure 23-13.

[2] See Figure 23-14.

[3] tk1 in the UDMA figures equals (tk -2*T)

23.4.1.3 Ultra DMA In Timing Diagrams

UDMA mode timing is more complicated than PIO mode or MDMA mode. In this section, timing diagrams for UDMA are provided.

Figure 23-15 shows timing for UDMA in transfer start.

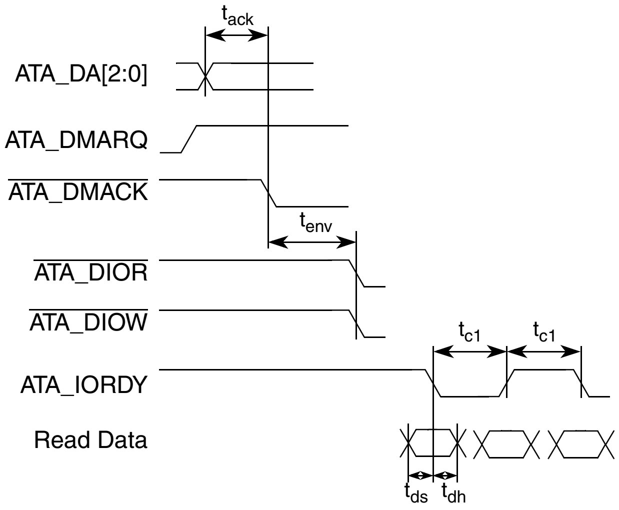


Figure 23-15. UDMA in Transfer Start Timing Diagram

Figure 23-16 shows timing for host terminating UDMA in transfer.

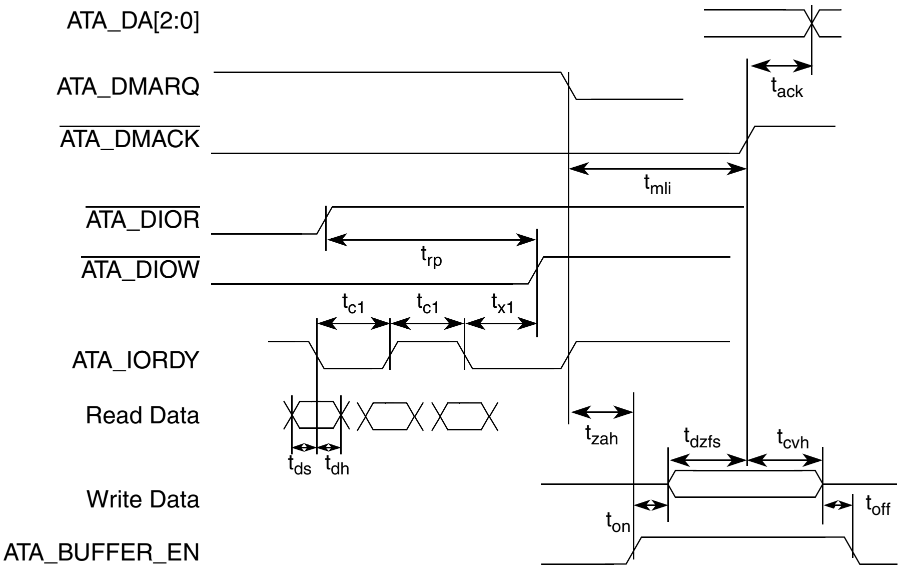


Figure 23-16. UDMA in Host Terminates Transfer

Figure 23-17 shows timing for device terminating UDMA in transfer.

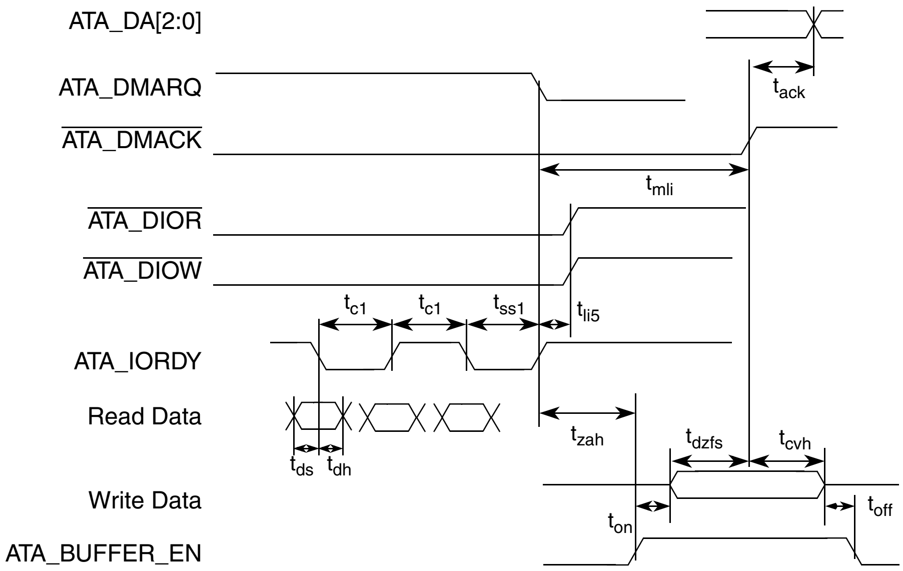


Figure 23-17. UDMA in Device Terminates Transfer

Timing parameters for UDMA in burst appear in Table 23-11.

Table 23-11. Timing Parameters for UDMA in Burst

ATA Parameter	Spec Parameter	Value	How to Meet?
tack	tack	tack(min) = (time_ack * T) - (tskew1 + tskew2)	TIME_ACK

tenv	tenv	tenv(min) = (time_env * T) - (tskew1 + tskew2)
		tenv(max) = (time_env * T) + (tskew1 + tskew2)
			TIME_ENV

tds	tds1	tds - (tskew3) - ti_ds > 0	TSKEW3, TI_DS, TI_DH should be low enough
tdh	tdh1	tdh - (tskew3) -ti_dh > 0	
tcyc	tc1	(tcyc - tskew + **TBD**) > T	T big enough
trp	trp	trp(min) = time_rp * T - (tskew1 + tskew2 + tskew6)	TIME_RP
	tx1[1]	(time_rp * T) - (tco + tsu + 3T + 2 *tbuf + 2*tcable2) > trfs (drive)	TIME_RP
tmli	tmli1	tmli1(min) = (time_mlix + 0.4) * T	TIME_MLIX
tzah	tzah	tzah(min) = (time_zah + 0.4) * T	TIME_ZAH
tdzfs	tdzfs	tdzfs = (time_dzfs * T) - (tskew1 + tskew2)	TIME_DZFS
tcvh	tcvh	tcvh = (time_cvh *T) - (tskew1 + tskew2)	TIME_CVH

	ton
	toff
		ton = time_on * T - tskew1
		toff = time_off * T - tskew1	

[1] There is a special timing requirement in the ATA host requiring the internal DIOW to only go high three clocks after the last active edge on the DSTROBE signal. The equation on this line tries to capture this constraint.

2. Make ton and toff big enough to avoid bus contention.

23.4.1.4 Ultra DMA Out Timing Diagrams

UDMA mode timing is more complicated than PIO mode or MDMA mode. In this section, timing diagrams for UDMA out are provided. Figure 23-18 shows timing for UDMA out transfer start.

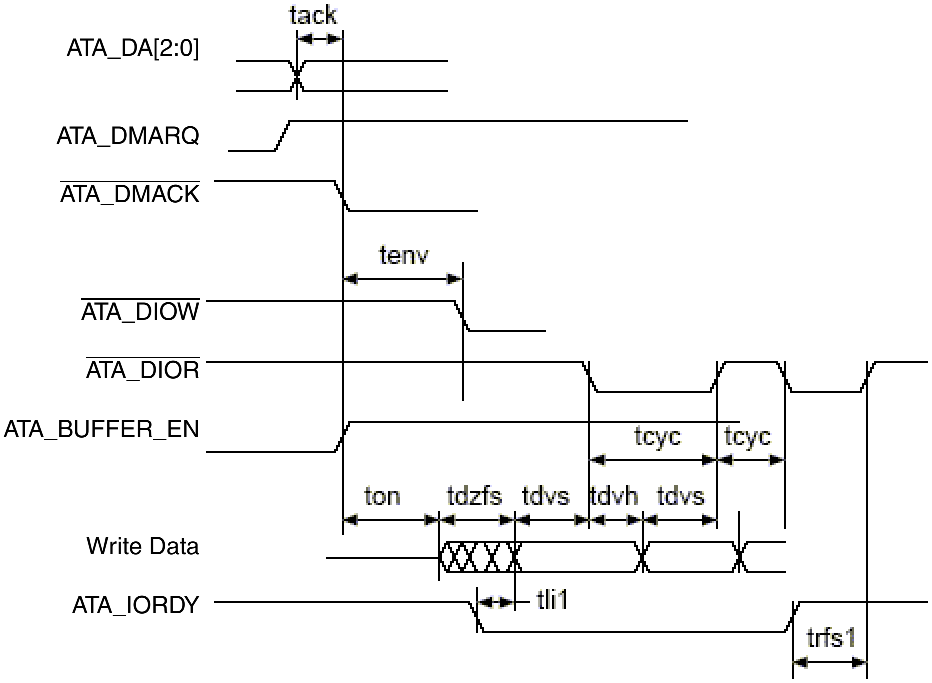


Figure 23-18. UDMA Out Transfer Start Timing Diagram

Figure 23-19 shows timing for host terminating UDMA out transfer.

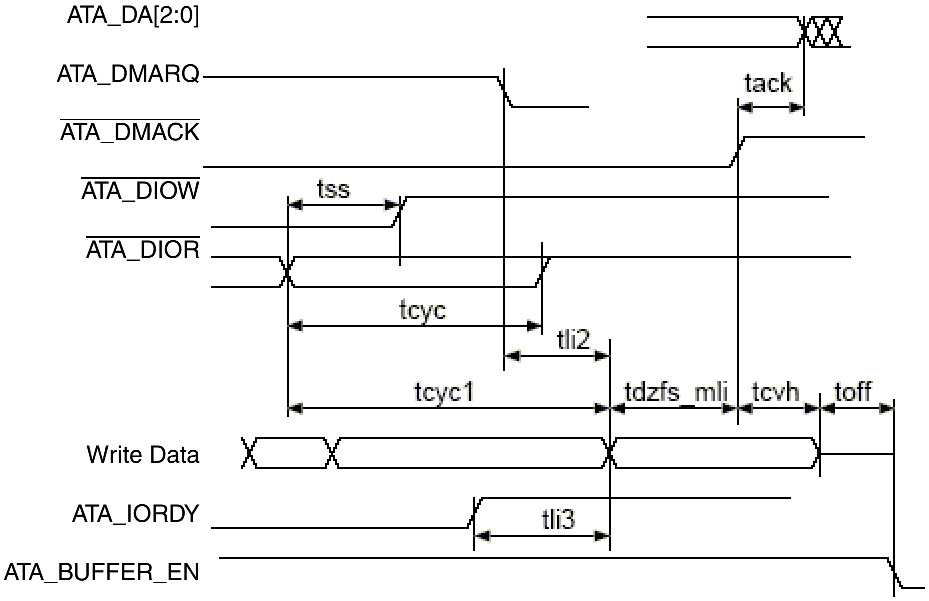


Figure 23-19. UDMA Out Host Terminates Transfer

Figure 23-20 shows timing for device terminating UDMA out transfer.

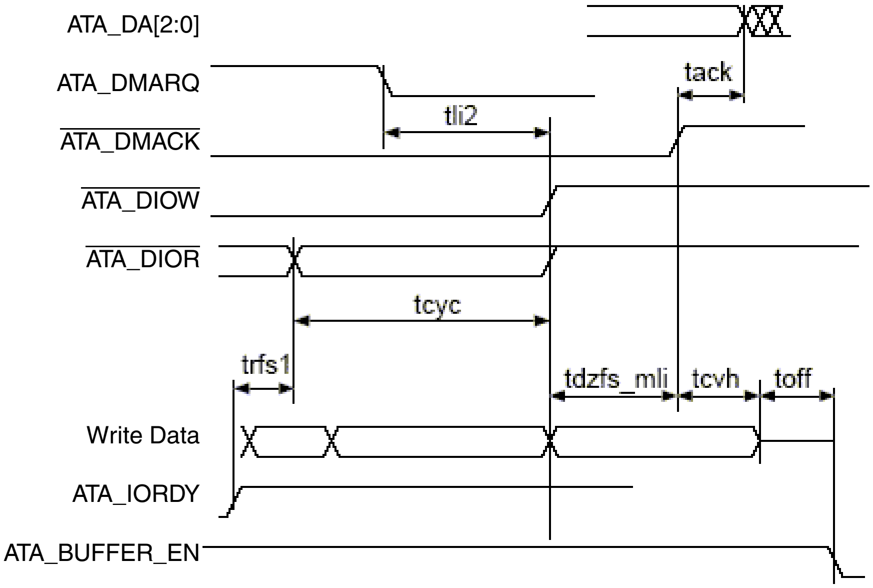


Figure 23-20. UDMA Out Device Terminates Transfer

Timing parameters for UDMA out burst appear in Table 23-12.

Table 23-12. Timing Parameters UDMA Out Burst

ATA Parameter	Spec Parameter	Value	How to Meet?
tack	tack	tack(min) = (time_ack * T) - (tskew1 + tskew2)	TIME_ACK

tenv	tenv	tenv(min) = (time_env * T) - (tskew1 + tskew2)
		tenv(max) = (time_env * T) + (tskew1 + tskew2)
			TIME_ENV

tdvs	tdvs	tdvs = (time_dvs * T) - (tskew1 + tskew2)	TIME_DVS
tdvh	tdvh	tdvs = (time_dvh * T) - (tskew1 + tskew2)	TIME_DVH
tcyc	tcyc	tcyc = time_cyc * T - (tskew1 + tskew2)	TIME_CYC
t2cyc		t2cyc = time_cyc * 2 * T	TIME_CYC
trfs1	trfs	trfs = 1.6 * T + tsui + tco + tbuf + tbuf	—
—	tdzfs	tdzfs = time_dzfs * T - (tskew1)	TIME_DZFS
tss	tss	tss = time_ss * T - (tskew1 + tskew2)	TIME_SS
tmli	tdzfs_mli	tdzfs_mli =max(time_dzfs, time_mli) * T - (tskew1 + tskew2)	
tli	tli1	tli1 > 0	
tli	tli2	tli2 > 0	
tli	tli3	tli3 > 0	
tcvh	tcvh	tcvh = (time_cvh *T) - (tskew1 + tskew2)	TIME_CVH

	ton
	toff
		ton = time_on * T - tskew1
		toff = time_off * T - tskew1	

23.4.2 Resetting ATA Bus

The ATA bus reset, $\overline{\text{ATA\_RESET}}$, asserts when the ATA_CR[RESET] bit is cleared. At the same time, the ATA protocol engine is reset. When this bit is set, the reset releases.

23.4.3 Programming ATA Bus Timing and IORDYEN

The timing ATA interface operates with on the ATA bus is programmable with the 24 timing registers. How these registers affect the timing parameters on the ATA bus, is detailed in Section 23.4.1, “Timing on ATA Bus.” The user can reprogram these registers at any time when the ATA bus is idle, so before reprogramming make sure that:

- ATA_CR[DMAPEND] bit is cleared and
- ATA_ISR[IDLE] bit is set.

These two conditions can be accomplished by first clearing the ATA_CR[DMAPEND] bit, then waiting until the ATA_ISR[IDLE] bit is set. If ATA_CR[DMAPEND] was set before the reprogramming starts, it should be set again after new timing is in effect to allow the drive to finish the current DMA transfer.

Only reprogram the bus timing in the middle of an ongoing DMA transfer when necessary because the operating system wants to change the bus clock period. (Dynamic voltage frequency scaling).

Wait for ATA_ISR[IDLE] because a PIO read or write to the ATA bus terminates after the bus cycle with the CPU terminates. If the wait for ATA_ISR[IDLE] does not occur, the new timing values may affect a running bus cycle and cause error.

The ATA_CR[IORDYEN] bit influences whether the ATA interface responds to the iordy signal coming from the drive. To reprogram it the same rules as the timing registers apply: Only allowed when ATA_CR[DMAPEND] is cleared, while ATA_ISR[IDLE] is set.

23.4.4 Access to ATA Bus in PIO Mode

Access to the ATA bus in PIO mode is possible after:

- ATA_CR[RESET] bit is set.
- Timing parameters are programmed.

To access the drive in PIO mode, simply read or write to the correct drive register. The bus cycle is translated to an ATA cycle, and the drive is accessed.

When drive registers are accessed while the ATA bus is in reset, the read or write is discarded, not done.

23.4.5 Using DMA Mode to Receive Data from ATA Bus

Apart from PIO mode, the ATA interface also supports MDMA and UDMA mode to transfer data. DMA mode can receive data from the drive (DMA in transfer). In DMA receive mode, the protocol engine transfers data from the drive to the FIFO using multiword DMA or ultra DMA protocol. The transfer pauses when:

- The FIFO is full.

- The drive deasserts its DMA request signal, ATA_DMARQ.
- The ATA_CR[DMAPEND] bit is cleared.

When the cause of the transfer pausing is removed, the transfer restarts. The end of the transfer is signalled by the drive to the host by asserting the ATA_INTRQ signal. Alternatively, the host can read the device status register. In this register, the drive also indicates if the transfer has ended.

The host system DMA manages the transfer of data from FIFO into the memory. When the FIFO filling is above the alarm threshold, the DMA should read one packet of data from the FIFO and store this in main memory. In doing so, the DMA prevents the FIFO from getting full and keeps the transfer from drive to FIFO running.

The steps for setting up a DMA data transfer from device to host are:

1. Make sure the ATA bus is not in reset and all timing registers are programmed.
2. Make sure the FIFO is empty by reading it until empty or by resetting it.
3. Initialize the DMA channel associated with ATA receive. Every time ATA receive DMA request is asserted, the DMA should read <packetsize> longwords from the FIFO and store them to main memory. (typical packetsize is eight longwords)
4. Write 2 * <packetsize> to the FIFO_ALARM register. In this way, the FIFO requests attention to DMA when there is at least one packet ready for transfer.
5. To make the ATA ready for a DMA transfer from device to host, take the following steps:
   a) Make sure the FIFO is enabled by setting the ATACR[FEN] bit.
   b) Set the ATA_CR[FEMPTY] bit which enables the FIFO to by emptied by the DMA.
   c) Program ATA_CR[DMAPEND] equals 1 and ATA_CR[DMADIR] equals 0. Also, select the DMA mode; ATA_CR[DMAMODE] equals 0 for multiword DMA or ATA_CR[DMAMODE] equals 1 for ultra DMA.
6. Now, the host side of the DMA is ready. Send commands to the drive in PIO mode causing it to request DMA transfer on the ATA bus. The nature of these commands is beyond the scope of this document. Consult the ATA specification for more information on how to communicate with the drive.
7. When the drive now requests a DMA transfer by asserting ATA_DMARQ, the ATA interface acknowledges with ATA_DMACK and transfer starts. Data transfers automatically to the FIFO and to the host memory from there.
8. During the transfer, the host can monitor for end-of-transfer by reading some device ATA registers. These reads cause the running DMA to pause; after the read completes and the DMA resumes. The host can also wait unit the drive asserts ATA_INTRQ. This also indicates end-of-transfer.
9. On end-of-transfer, the host or host DMA should wait until ATA_ISR[IDLE] is set; next read, the remaining halfwords from the FIFO and transfer these to memory.

NOTE

There may be less than <packetsize> remaining bytes, so the transfer is not automatic by the DMA.

23.4.6 Using DMA Mode to Transmit Data to ATA Bus

Apart from PIO mode, the ATA interface supports also multiword DMA (MDMA) and ultra DMA (UDMA) mode to transfer data. DMA mode can transmit data to the drive (DMA out transfer). In DMA transmit mode, the protocol engine transfers data from the FIFO to the drive using multiword DMA or ultra DMA protocol. The transfer pauses when:

- The FIFO is empty.
- The drive deasserts its DMA request signal, ATA_DMARQ.
- The ATA_CR[DMAPEND] bit is cleared.

When the cause of transfer pausing is removed, the transfer restarts. The end of transfer is signalled by the drive to the host by asserting the ATA_INTRQ signal. Alternatively, the host can read the device status register. In this register, the drive also indicate if the transfer has ended.

The host system DMA manages the transfer of data from the memory to the FIFO. When the FIFO filling is below the alarm threshold, the DMA must read one packet of data from the main memory and store this in the FIFO. In doing so, the DMA prevents the FIFO from becoming empty and keeps the transfer from FIFO to drive running.

The steps for setting up a DMA data transfer from device to host are:

1. Make sure the ATA bus is not in reset and all timing registers are programmed.
2. Make sure the FIFO is empty by reading it until empty or by resetting it.
3. Initialize the DMA channel associated with ATA transmit. Every time ATA transmit DMA request asserts, the DMA should read <packetsize> longwords from the main memory, and write them to the FIFO. (typical packetsize is 8 longwords). Program the DMA so it does not transfer more than <sectorsize> longwords in total.
4. Write FIFO_SIZE - 2 * <packetsize> to the FIFO_ALARM register. In this way, the FIFO requests attention to DMA when there is room for at least one extra packet. FIFO_SIZE should be given in halfwords. (typical 64 halfwords)
5. To make the ATA ready for a DMA transfer from host to device, perform the following steps:
   a) Make sure the FIFO is enabled by setting the ATA_CR[FEN] bit.
   b) Set the ATA_CR[FREFILL] bit, which enables the DMA to fill FIFO.
   c) Program ATA_CR[DMAPEND] equals 1 and ATA_CR[DMADIR] equals 1. Also, select the DMA mode; ATA_CR[DMAMODE] equals 0 for multiword DMA or ATA_CR[DMAMODE] equals 1 for ultra DMA.
6. The host side of the DMA is now ready. Send commands to the drive in PIO mode that cause it to request DMA transfer on the ATA bus. The nature of these commands is beyond the scope of this document. Consult the ATA specification for more information on how to communicate with the drive.
7. When the drive now requests DMA transfer by asserting ATA_DMARQ, the ATA interface acknowledges with $\overline{\text{ATA\_DMACK}}$, and the transfer starts. Data is transferred automatically from the FIFO, and also from host memory to FIFO.

8. During the transfer, the host can monitor for end-of-transfer by reading some device ATA registers. These reads cause the running DMA to pause; after the read completes, the DMA resumes. The host can also wait until the drive asserts ATA_INTRQ. This also indicates end of transfer.

On end-of-transfer, no extra FIFO manipulations are needed.

Chapter 24
Cryptographic Acceleration Unit (CAU)

24.1 Introduction

The cryptographic acceleration unit (CAU) is a ColdFire coprocessor implementing a set of specialized operations in hardware to increase the throughput of software-based encryption and hashing functions.

24.1.1 Block Diagram

Figure 24-1 shows a simplified block diagram of the CAU.

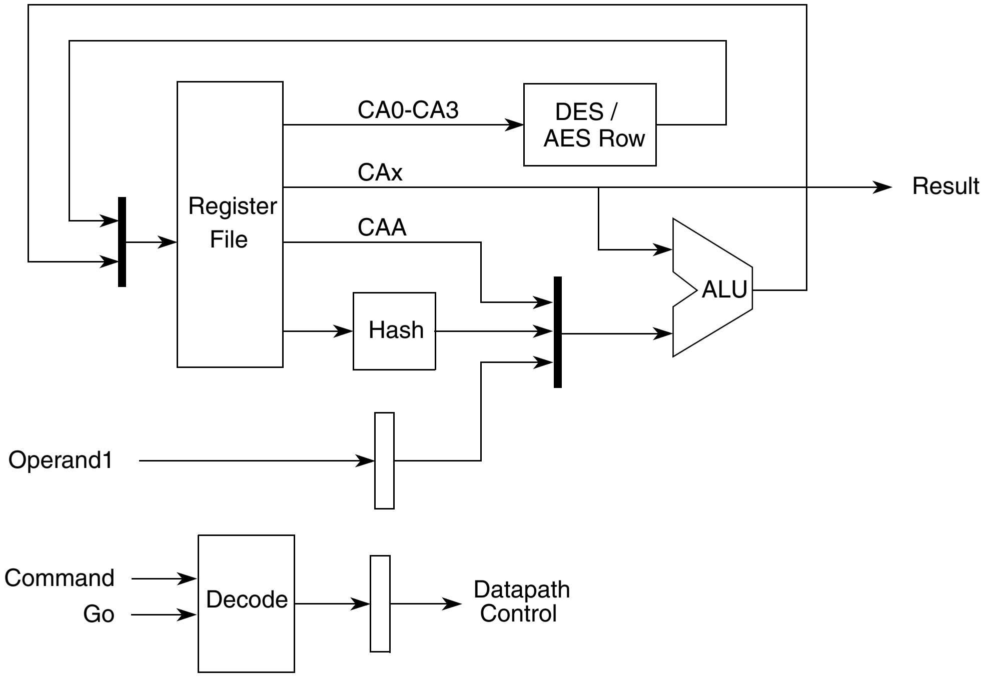


Figure 24-1. Top Level CAU Block Diagram

24.1.2 Overview

The CAU supports acceleration of the following algorithms:

- DES
- 3DES
- AES
- MD5

- SHA-1

This selection of algorithms provides excellent support for network security standards (SSL, IPsec). Additionally, using the CAU efficiently permits the implementation of any higher level functions or modes of operation (HMAC, CBC, etc.) based on the supported algorithm.

The CAU is an instruction-level ColdFire coprocessor. The cryptographic algorithms are implemented partially in software with only functions critical to increasing performance implemented in hardware. The ColdFire coprocessor allows for efficient, fine-grained partitioning of functions between hardware and software.

- Implement the innermost round functions by using the coprocessor instructions
- Implement higher-level functions in software by using the standard ColdFire instructions

This partitioning of functions is key to minimizing size of the CAU while maintaining a high level of throughput. Using software for some functions also simplifies the CAU design. The CAU implements a set of 22 coprocessor commands that operate on a register file of eight 32-bit registers. It is tightly coupled to the ColdFire core and there is no local memory or external interface.

24.1.3 Features

The CAU includes these distinctive features:

- Supports DES, 3DES, AES, MD5, SHA-1 algorithms
- Simple, flexible programming model

24.2 Memory Map/Register Definition

The CAU only supports longword operations and register accesses. All registers support read, write, and ALU operations. However, only bits 1–0 of the CASR are writeable. Bits 31–2 of the CASR must be written as 0 for compatibility with future versions of the CAU.

Table 24-1. CAU Memory Map

Code	Register	DES	AES	SHA-1	MD5	Access	Reset Value	Section/Page
0	CAU status register (CASR)	—	—	—	—	R/W	0x1000_0000	24.2.1/24-4
1	CAU accumulator (CAA)	—	—	T	a	R	0x0000_0000	24.2.2/24-4
2	General purpose register 0 (CA0)	C	W0	A	—	R	0x0000_0000	24.2.3/24-5
3	General purpose register 1 (CA1)	D	W1	B	b	R	0x0000_0000	24.2.3/24-5
4	General purpose register 2 (CA2)	L	W2	C	c	R	0x0000_0000	24.2.3/24-5
5	General purpose register 3 (CA3)	R	W3	D	d	R	0x0000_0000	24.2.3/24-5
6	General purpose register 4 (CA4)	—	—	E	—	R	0x0000_0000	24.2.3/24-5
7	General purpose register 5 (CA5)	—	—	W	—	R	0x0000_0000	24.2.3/24-5

24.2.1 CAU Status Register (CASR)

CASR contains the status and configuration for the CAU.

Register code: 0x0 (CASR)

Access: Read/write via CAU commands

	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
R	VER				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DPE	IC
W																																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-2. CAU Status Register (CASR)

Table 24-2. CASR Field Descriptions

Field	Description
31–28 VER	CAU version. Indicates CAU version 0x1 Initial CAU version (This is the value on this device) 0x2 Second version, added support for SHA-256 algorithm
27–2	Reserved, must be cleared.
1 DPE	DES parity error. 0 No error detected 1 DES key parity error detected
0 IC	Illegal command. Indicates an illegal instruction not found in Section 24.3.3, “CAU Commands,” has been executed. 0 No illegal commands issued 1 Illegal coprocessor command issued

24.2.2 CAU Accumulator (CAA)

CAU commands use the CAU accumulator for storage of results and as an operand for the cryptographic algorithms.

Register code: 0x1 (CAA)

Access: Read/write via CAU commands

	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
R W	ACC																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-3. CAU Accumulator Register (CAA)

Table 24-3. CAA Field Descriptions

Field	Description
31–0 ACC	Accumulator. Stores results of various CAU commands.

24.2.3 CAU General Purpose Registers (CA*n*)

The nine CAU general purpose registers are used in the CAU commands for storage of results and as operands for the various cryptographic algorithms.

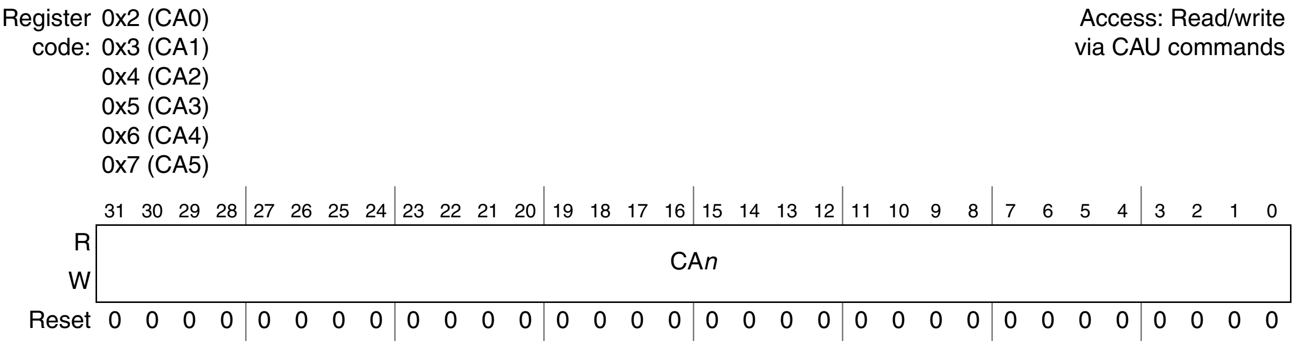


Figure 24-4. CAU General Purpose Registers (CA*n*)

Table 24-4. CA*n* Field Descriptions

Field	Description
31–0 CA*n*	General purpose registers. Used by the CAU commands. Some cryptographic operations work with specific registers.

24.3 Functional Description

24.3.1 Programming Model

The CAU is an instruction-level coprocessor. It has a dedicated register file, a specialized ALU, and specialized units for performing cryptographic operations. The CAU design uses a simple, flexible accumulator-based architecture. Most commands, including load and store, can specify any register in the register file. Some cryptographic operations work with specific registers.

24.3.2 Coprocessor Instructions

Operation of the CAU is controlled via standard ColdFire coprocessor load (cp0ld) and store (cp0st) instructions. The CAU has a dedicated register file accessed using these instructions. The load instruction loads CAU registers and specifies CAU operations. The store instruction stores CAU registers. The example assembler syntax for the CAU is:

```
cp0ld.l          <ea>,<CMD>       ; coprocessor load
cp0st.l          <ea>,<CMD>       ; coprocessor store
```

The <ea> field specifies the source operand (operand1) for load instructions and destination (result) for store instructions. The basic ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)} are supported for this field. The <CMD> field is a 9-bit value that specifies the CAU command for an instruction. Table 24-5 shows how the CAU supports a single command (STR) for store instructions and 21 commands for the load instructions. The CAU only supports longword operations. A CAU command can be issued every clock cycle.

24.3.3 CAU Commands

The CAU supports the commands shown in Table 24-5. All other encodings are reserved. The CASR[IC] bit is set if an undefined command is issued. A specific illegal command (ILL) is defined to allow software self-checking. Reserved commands should not be issued to ensure compatibility with future implementations.

The CMD field specifies the CAU command for the instruction.

Table 24-5. CAU Commands

Inst Type	Command Name	Description	CMD									Operation
			8	7	6	5	4	3	2	1	0	
cp0ld	CNOP	No Operation	0x000									—
cp0ld	LDR	Load Reg	0x01					CAx				Op1 → CAx
cp0st	STR	Store Reg	0x02					CAx				CAx → Result
cp0ld	ADR	Add	0x03					CAx				CAx + Op1 → CAx
cp0ld	RADR	Reverse and Add	0x04					CAx				CAx + ByteRev(Op1) → CAx
cp0ld	ADRA	Add Reg to Acc	0x05					CAx				CAx + CAA → CAA
cp0ld	XOR	Exclusive Or	0x06					CAx				CAx ^ Op1 → CAx
cp0ld	ROTL	Rotate Left	0x07					CAx				CAx <<< Op1 → CAx
cp0ld	MVRA	Move Reg to Acc	0x08					CAx				CAx → CAA
cp0ld	MVAR	Move Acc to Reg	0x09					CAx				CAA → CAx
cp0ld	AESS	AES Sub Bytes	0x0A					CAx				SubBytes(CAx) → CAx
cp0ld	AESIS	AES Inv Sub Bytes	0x0B					CAx				InvSubBytes(CAx) → CAx
cp0ld	AESC	AES Column Op	0x0C					CAx				MixColumns(CAx)^Op1→ CAx
cp0ld	AESIC	AES Inv Column Op	0x0D					CAx				InvMixColumns(CAx^Op1) → CAx
cp0ld	AESR	AES Shift Rows	0x0E0									ShiftRows(CA0-CA3) → CA0-CA3
cp0ld	AESIR	AES Inv Shift Rows	0x0F0									InvShiftRows(CA0-CA3)→ CA0-CA3
cp0ld	DESR	DES Round	0x10					IP	FP	KS[1:0]		DES Round(CA0-CA3)→CA0-CA3
cp0ld	DESK	DES Key Setup	0x11					0	0	CP	DC	DES Key Op(CA0-CA1)→CA0-CA1 Key Parity Error & CP → CASR[1]
cp0ld	HASH	Hash Function	0x12					0	HF[2:0]			Hash Func(CA1-CA3)+CAA→CAA
cp0ld	SHS	Secure Hash Shift	0x130									CAA <<< 5→ CAA, CAA→CA0, CA0→CA1, CA1 <<< 30 → CA2, CA2→CA3, CA3→CA4
cp0ld	MDS	Message Digest Shift	0x140									CA3→CAA, CAA→CA1, CA1→CA2, CA2→CA3,
cp0ld	ILL	Illegal Command	0x1F0									0x1→CASR[0]

Section 24.4.2, “Assembler Equate Values,” contains a set of assembly constants used in the command descriptions here. If supported by the assembler, macros can also be created for each instruction. The value CA*x* should be interpreted as any CAU register (CASR, CAA, CA*n*) and the <ea> field as one of the supported ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)}. For example, the instruction to add the value from the core register D1 to the CAU register CA0 is:

```
cp0ld.l           %d1,#ADR+CA0          ; CA0=CA0+d1
```

24.3.3.1 Coprocessor No Operation (CNOP)

```
cp0ld.l    #CNOP
```

The CNOP command is the coprocessor no-op defined by the ColdFire coprocessor definition for synchronization. It is not actually issued to the coprocessor from the core.

24.3.3.2 Load Register (LDR)

```
cp0ld.l    <ea>,#LDR+CAx
```

The LDR command loads CAx with the source data specified by <ea>.

24.3.3.3 Store Register (STR)

```
cp0st.l    <ea>,#STR+CAx
```

The STR command stores the value from CAx to the destination specified by <ea>.

24.3.3.4 Add to Register (ADR)

```
cp0ld.l    <ea>,#ADR+CAx
```

The ADR command adds the source operand specified by <ea> to CAx and stores the result in CAx.

24.3.3.5 Reverse and Add to Register (RADR)

```
cp0ld.l    <ea>,#RADR+CAx
```

The RADR command performs a byte reverse on the source operand specified by <ea>, adds that value to CAx, and stores the result in CAx. Table 24-6 shows an example.

Table 24-6. RADR Command Example

Operand	CAx Before	CAx After
0x0102_0304	0xA0B0_C0D0	0xA4B3_C2D1

24.3.3.6 Add Register to Accumulator (ADRA)

```
cp0ld.l    #ADRA+CAx
```

The ADRA command adds CAx to CAA and stores the result in CAA.

24.3.3.7 Exclusive Or (XOR)

```
cp0ld.l    <ea>,#XOR+CAx
```

The XOR command performs an exclusive-or of the source operand specified by <ea> with CAx and stores the result in CAx.

24.3.3.8 Rotate Left (ROTL)

```
cp0ld.l    <ea>,#ROTL+CAx
```

ROTL rotates the CAx bits to the left with the result stored back to CAx. The number of bits to rotate is the value specified by <ea> modulo 32.

24.3.3.9 Move Register to Accumulator (MVRA)

```
cp0ld.l    #MVRA+CAx
```

The MVRA command moves the value from the source register CAx to the destination register CAA.

24.3.3.10 Move Accumulator to Register (MVAR)

```
cp0ld.l    #MVAR+CAx
```

The MVAR command moves the value from source register CAA to the destination register CAx.

24.3.3.11 AES Substitution (AESS)

```
cp0ld.l    #AESS+CAx
```

The AESS command performs the AES byte substitution operation on CAx and stores the result back to CAx.

24.3.3.12 AES Inverse Substitution (AESIS)

```
cp0ld.l    #AESIS+CAx
```

The AESIS command performs the AES inverse byte substitution operation on CAx and stores the result back to CAx.

24.3.3.13 AES Column Operation (AESC)

```
cp0ld.l    <ea>,#AESC+CAx
```

The AESC command performs the AES column operation on the contents of CAx then performs an exclusive-or of that result with the source operand specified by <ea> and stores the result in CAx.

24.3.3.14 AES Inverse Column Operation (AESIC)

```
cp0ld.l    <ea>,#AESIC+CAx
```

The AESIC command performs an exclusive-or operation of the source operand specified by <ea> on the contents of CAx followed by the AES inverse mix column operation on that result and stores the result back in CAx.

24.3.3.15 AES Shift Rows (AESR)

```
cp0ld.l    #AESR
```

The AESR command performs the AES shift rows operation on registers CA0, CA1, CA2, and CA3. Table 24-7 shows an example.

Table 24-7. AESR Command Example

Register	Before	After
CA0	0x0102_0304	0x0106_0B00
CA1	0x0506_0708	0x050A_0F04
CA2	0x090A_0B0C	0x090E_0308
CA3	0x0D0E_0F00	0x0D02_070C

24.3.3.16 AES Inverse Shift Rows (AESIR)

```
cp0ld.l    #AESIR
```

The AESIR command performs the AES inverse shift rows operation on registers CA0, CA1, CA2 and CA3. Table 24-8 has an example.

Table 24-8. AESIR Command Example

Register	Before	After
CA0	01060B00	01020304
CA1	050A0F04	05060708
CA2	090E0308	090A0B0C
CA3	0D02070C	0D0E0F00

24.3.3.17 DES Round (DESR)

```
cp0ld.l    #DESR+{IP}+{FP}+{KSx}
```

The DESR command performs a round of the DES algorithm and a key schedule update with the following source and destination designations: CA0=C, CA1=D, CA2=L, CA3=R. If the IP bit is set, DES initial permutation performs on CA2 and CA3 before the round operation. If the FP bit is set, DES final permutation (inverse initial permutation) performs on CA2 and CA3 after the round operation. The round operation uses the source values from registers CA0 and CA1 for the key addition operation. The KSx field specifies the shift for the key schedule operation to update the values in CA0 and CA1. Table 24-9 defines the specific shift function performed based on the KSx field.

Table 24-9. Key Shift Function Codes

KSx Code	KSx Define	Shift Function
0	KSL1	Left 1
1	KSL2	Left 2

Table 24-9. Key Shift Function Codes (continued)

KSx Code	KSx Define	Shift Function
2	KSR1	Right 1
3	KSR2	Right 2

24.3.3.18 DES Key Setup (DESK)

```
        cp0ld.l    #DESK+{CP}+{DC}
```

The DESK command performs the initial key transformation (permuted choice 1) defined by the DES algorithm on CA0 and CA1 with CA0 containing bits 1–32 of the key and CA1 containing bits 33–64 of the key[1]. If the DC bit is set, no shift operation performs and the values C_0 and D_0 store back to CA0 and CA1 respectively. The DC bit should be set for decrypt operations. If the DC bit is not set, a left shift by one also occurs and the values C_1 and D_1 store back to CA0 and CA1 respectively. The DC bit should be cleared for encrypt operations. If the CP bit is set and a key parity error is detected, CASR[DPE] bit is set; otherwise, it is cleared.

24.3.3.19 Hash Function (HASH)

```
        cp0ld.l    #HASH+HFx
```

The HASH command performs a hashing operation on a set of registers and adds that result to the value in CAA and stores the result in CAA. The specific hash function performed is based on the HFx field as defined in Table 24-10.

Table 24-10. Hash Function Codes

HFx Code	HFx Define	Hash Function	Hash Logic
0	HFF	MD5 F()	(CA1 & CA2) | ($\overline{\text{CA1}}$ & CA3)
1	HFG	MD5 G()	(CA1 & CA3) | (CA2 & $\overline{\text{CA3}}$)
2	HFH	MD5 H(), SHA Parity()	CA1 ^ CA2 ^ CA3
3	HFI	MD5 I()	CA2 ^ (CA1 | $\overline{\text{CA3}}$)
4	HFC	SHA Ch()	(CA1 & CA2) ^ ($\overline{\text{CA1}}$ & CA3)
5	HFM	SHA Maj()	(CA1 & CA2) ^ (CA1 & CA3) ^ (CA2 & CA3)

24.3.3.20 Secure Hash Shift (SHS)

```
        cp0ld.l    #SHS
```

The SHS command does a set of parallel register-to-register move and shift operations for implementing SHA-1. The following source and destination assignments are made: CAA=CAA<<<5, CA0=CAA, CA1=CA0, CA2=CA1<<<30, CA3=CA2, CA4=CA3.

[1] The DES algorithm numbers the most significant bit of a block as bit 1 and the least significant as bit 64.

24.3.3.21 Message Digest Shift (MDS)

```
cp0ld.l   #MDS
```

The MDS command does a set of parallel register-to-register move operations for implementing MD5. The following source and destination assignments are made: CAA=CA3, CA1=CAA, CA2=CA1, CA3=CA2.

24.3.3.22 Illegal Command (ILL)

```
cp0ld.l   #ILL
```

The ILL command is a specific illegal command that sets CASR[IC]. All other illegal commands are reserved for use in future implementations.

24.4 Application/Initialization Information

24.4.1 Code Example

A code fragment is shown below as an example of how the CAU is used. This example shows the round function of the AES algorithm. Core register A0 is pointing to the key schedule.

```
cp0ld.l   #AESS+CA0          ; sub bytes w0
cp0ld.l   #AESS+CA1          ; sub bytes w1
cp0ld.l   #AESS+CA2          ; sub bytes w2
cp0ld.l   #AESS+CA3          ; sub bytes w3
cp0ld.l   #AESR              ; shift rows
cp0ld.l   (%a0)+,#AESC+CA0   ; mix col, add key w0
cp0ld.l   (%a0)+,#AESC+CA1   ; mix col, add key w1
cp0ld.l   (%a0)+,#AESC+CA2   ; mix col, add key w2
cp0ld.l   (%a0)+,#AESC+CA3   ; mix col, add key w3
```

24.4.2 Assembler Equate Values

The following equates ease programming of the CAU.

```
; CAU Registers (CAx)
         .set     CASR,0x0
         .set     CAA,0x1
         .set     CA0,0x2
         .set     CA1,0x3
         .set     CA2,0x4
         .set     CA3,0x5
         .set     CA4,0x6
         .set     CA5,0x7

; CAU Commands
         .set     CNOP,0x000
         .set     LDR,0x010
         .set     STR,0x020
         .set     ADR,0x030
         .set     RADR,0x040
         .set     ADRA,0x050
         .set     XOR,0x060
         .set     ROTL,0x070
         .set     MVRA,0x080
```

```
        .set     MVAR,0x090
        .set     AESS,0x0A0
        .set     AESIS,0x0B0
        .set     AESC,0x0C0
        .set     AESIC,0x0D0
        .set     AESR,0x0E0
        .set     AESIR,0x0F0
        .set     DESR,0x100
        .set     DESK,0x110
        .set     HASH,0x120
        .set     SHS,0x130
        .set     MDS,0x140
        .set     ILL,0x1F0

; DESR  Fields
        .set     IP,0x08            ; initial permutation
        .set     FP,0x04            ; final permutation
        .set     KSL1,0x00          ; key schedule left 1 bit
        .set     KSL2,0x01          ; key schedule left 2 bits
        .set     KSR1,0x02          ; key schedule right 1 bit
        .set     KSR2,0x03          ; key schedule right 2 bits

; DESK Field
        .set     DC,0x01            ; decrypt key schedule
        .set     CP,0x02            ; check parity

; HASH Functions Codes
        .set     HFF,0x0            ; MD5 F() CA1&CA2 | ~CA1&CA3
        .set     HFG,0x1            ; MD5 G() CA1&CA3 | CA2&~CA3
        .set     HFH,0x2            ; MD5 H(), SHA Parity() CA1^CA2^CA3
        .set     HFI,0x3            ; MD5 I()  CA2^(CA1|~CA3)
        .set     HFC,0x4            ; SHA Ch() CA1&CA2 ^ ~CA1&CA3
        .set     HFM,0x5            ; SHA Maj() CA1&CA2 ^ CA1&CA3 ^ CA2&CA3
```

Chapter 25
Random Number Generator (RNG)

25.1 Introduction

This chapter describes the random number generator (RNG), including a programming model, functional description, and application information.

NOTE

> The MCF54450, MCF54452, and MCF54454 do not contain cryptography modules. Refer to Table 1-1 for details on device configurations.

25.1.1 Overview

The random number generator (RNG) module is capable of generating 32-bit random numbers. It complies with Federal Information Processing Standard (FIPS) 140 standards for randomness and non-determinism. The random bits generate by clocking shift registers with clocks derived from ring oscillators. The configuration of the shift registers ensures statistically good data (data that looks random). The oscillators with their unknown frequencies provide the required entropy needed to create random data.

CAUTION

> There is no known cryptographic proof showing that this is a secure method of generating random data. In fact, there may be an attack against the random number generator if its output is used directly in a cryptographic application (the attack is based on the linearity of the internal shift registers). In light of this, it is highly recommended to use the random data produced by this module as an input seed to a NIST-approved (based on DES or SHA-1) or cryptographically-secure (RSA generator or BBS generator) random number generation algorithm.
>
> It is also recommended to use other sources of entropy along with the RNG to generate the seed to the pseudorandom algorithm. The more random sources combined to create the seed the better. The following is a list of sources that can be easily combined with the output of this module.
>
> – Current time using highest precision possible
> – Mouse and keyboard motions (or equivalent if being used on a cell phone or PDA)
> – Other entropy supplied directly by the user

NOTE

See Appendix D of the NIST Special Publication 800-90 “Recommendation for Random Number Generation Using Deterministic Random Bit Generators” for more information: http://csrc.nist.gov

25.2 Memory Map/Register Definition

Table 25-1 shows the address map for the RNG module. Detailed register descriptions are found in the following section.

Table 25-1. RNG Block Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_4000	RNG Control Register (RNGCR)	32	R/W	0x0000_0000	25.2.1/25-2
0xFC0B_4004	RNG Status Register (RNGSR)	32	R	0x0010_0000	25.2.2/25-3
0xFC0B_4008	RNG Entropy Register (RNGER)	32	W	0x0000_0000	25.2.3/25-4
0xFC0B_400C	RNG Output FIFO (RNGOUT)	32	R	0x0000_0000	25.2.4/25-4

25.2.1 RNG Control Register (RNGCR)

Immediately following reset, the RNG begins generating entropy (random data) in its internal shift registers. Random data is not pushed to the output FIFO until after the RNGCR[GO] bit is set. After this, a random 32-bit word is pushed to the FIFO every 256 cycles. If the FIFO is full, no push occurs. The FIFO is kept as close to full as possible.

: 0xFC0B_4000 (RNGCR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SLM	0	IM	HA	GO
W																													CI			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-1. RNG Control Register (RNGCR)

Table 25-2. RNGCR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 SLM	Sleep mode. The RNGA can be placed in low power mode by setting this bit. When this bit is set, the oscillators are disabled.Clearing this bit causes the RNGA to exit sleep mode. The FIFO is not pushed while the RNGA is in sleep mode. 0 RNGA is not in sleep mode. 1 RNGA is in sleep mode.
3 CI	Clear interrupt. Writing a 1 to this bit clears the error interrupt and RNGSR[EI]. This bit is self-clearing, 0 Do not clear error interrupt. 1 Clear error interrupt.

Table 25-2. RNGCR Field Descriptions (continued)

Field	Description
2 IM	Interrupt mask. 0 Error interrupt enabled. 1 Error interrupt masked.
1 HA	High assurance. Notifies core when FIFO underflow has occurred (FIFO is read while empty). Enables the security violation bit in the RNGSR. Bit is sticky and only cleared by hardware reset. 0 Disable security violation notification. 1 Enable security violation notification.
0 GO	Go bit. Starts/stops random data from being generated. Bit is sticky and only cleared by hardware reset. 0 FIFO not loaded with random data. 1 FIFO loaded with random data.

25.2.2 RNG Status Register (RNGSR)

The RNGSR, shown in Figure 25-2, is a read only register which reflects the internal status of the RNG.

: 0xFC0B_4004 (RNGSR) Access: User 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
R	0	0	0	0	0	0	0	0	OFS								OFL								0	0	0	SLP	EI	FUF	LRS	SV
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-2. RNG Status Register (RNGSR)

Table 25-3. RNGSR Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–16 OFS	Output FIFO size. Indicates size of the output FIFO (16 words) and maximum possible value of RNGR[OFL].
15–8 OFL	Output FIFO level. Indicates current number of random words in the output FIFO. Determines if valid random data is available for reading from the FIFO without causing an underflow condition.
7–5	Reserved, must be cleared.
4 SLP	Sleep. This bit reflects whether the RNG is in sleep mode. When this bit is set, the RNGA is in sleep mode and the oscillator clocks are inactive. While in this mode, the FIFO is not loaded and the FIFO level does not increase. 0 RNGA is not in sleep mode. 1 RNGA is in sleep mode.
3 EI	Error interrupt. Signals a FIFO underflow. Reset by a write to RNGCR[CI] and not masked by RNGCR[IM]. 0 FIFO not read while empty. 1 FIFO read while empty.
2 FUF	FIFO underflow. Signals FIFO underflow. Reset by reading RNGSR. 0 FIFO not read while empty since last read of RNGSR. 1 FIFO read while empty since last read of RNGSR.

Table 25-3. RNGSR Field Descriptions (continued)

Field	Description
1 LRS	Last read status. Reflects status of most recent read of the FIFO. 0 During last read, FIFO was not empty. 1 During last read, FIFO was empty (underflow condition).
0 SV	Security violation. When enabled by RNGCR[HA], signals that a FIFO underflow has occurred. Bit is sticky and is only cleared by hardware reset. 0 No violation occurred or RNGCR[HA] is cleared. 1 Security violation (FIFO underflow) has occurred.

25.2.3 RNG Entropy Register (RNGER)

The RNGER is a write-only register which allows the user to insert entropy into the RNG. This register allows an external user to continually seed the RNG with externally generated random data. Although use of this register is recommended, it is optional. The RNGER can be written at any time during operation.

Each time the RNGER is written, the value updates the internal state of the RNG. The update is performed in such a way that the entropy in the RNG's internal state is preserved. Use of the RNGER can increase the entropy but never decrease it.

: 0xFC0B_4008 (RNGER) Access: User 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
R																																
W	ENT																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-3. RNG Entropy Register (RNGER)

25.2.4 RNG Output FIFO (RNGOUT)

The RNGOUT provides temporary storage for random data generated by the RNG. As long as the FIFO is not empty, a read of this address returns 32 bits of random data. If the FIFO is read when it is empty, RNGSR[EI, FUF, LRS] are set. If the interrupt is enabled in RNGCR, an interrupt is triggered to the interrupt controller. The RNGSR[OFL], described in Section 25.2.2, "RNG Status Register (RNGSR)," can be polled to monitor how many 32-bit words are currently resident in the FIFO. A new random word pushes into the FIFO every 256 clock cycles (as long as the FIFO is not full). It is very important to poll RNGSR[OFL] to make sure random values are present before reading from RNGOUT.

: 0xFC0B_400C (RNGOUT) Access: User 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
R	Random Output																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-4. RNGOUT

25.3 Functional Description

Figure 25-5 shows the RNG has three functional blocks: output FIFO, internal bus interface, and the RNG core/control logic blocks. The following sections describe these blocks in more detail.

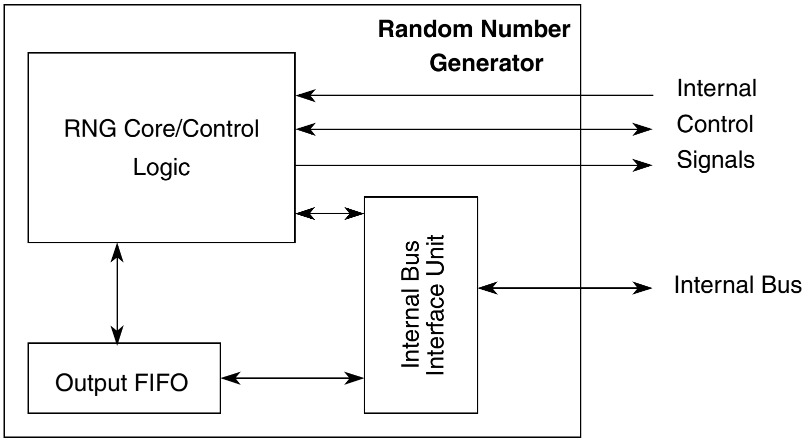


Figure 25-5. RNG Block Diagram

25.3.1 Output FIFO

The output FIFO provides temporary storage for random data that the RNG core/control logic block generates. This allows you to read multiple random long words back-to-back. The RNGSR allows the user to monitor the number of random words in the FIFO, through the output FIFO level field. If the user reads from the FIFO when it is empty and the interrupt is enabled, the RNG drives an interrupt request to the interrupt controller. It is very important to poll RNGSR[OFL] to make sure random values are present before reading from the FIFO.

25.3.2 RNG Core/Control Logic Block

This block contains the RNG's control logic as well as its core engine that generates random data.

25.3.2.1 RNG Control Block

The control block contains the address decoder, all addressable registers, and control state machines for the RNG. This block is responsible for communication with the peripheral interface and the FIFO interface. The block also controls the core engine to generate random data. The general functionality of the block is as follows. After reset, entropy generates and stores in the RNG's shift registers. After RNGCR[GO] is set, the FIFO is loaded with a random word every 256 cycles. The process of loading the FIFO continues as long as the FIFO is not full.

25.3.2.2 RNG Core Engine

The core engine block contains the logic that generates random data. The logic within the core engine contains the internal shift registers, as well as the logic that generates the two oscillator based clocks. This logic is brainless and must be controlled by the control block. The control block controls how the shift registers are configured and when the oscillator clocks are turned on.

25.4 Initialization/Application Information

The intended general operation of the RNG is as follows:

1. Reset/initialize.
2. Write to the RNG entropy register (optional).
3. Write to the RNG control register and set the interrupt mask, high assurance, and GO bits.
4. Poll RNGSR[OFL] to check for random data in the FIFO.
5. Read available random data from RNGOUT.
6. Repeat steps 3 and 4 as needed.

Chapter 26
Fast Ethernet Controllers (FEC0 and FEC1)

26.1 Introduction

This chapter provides a -set overview, a functional block diagram, and transceiver connection information for the 10 and 100 Mbps MII (media independent interface), as well as the low pin-count 10/100 Mbps reduced MII and 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

26.1.1 Overview

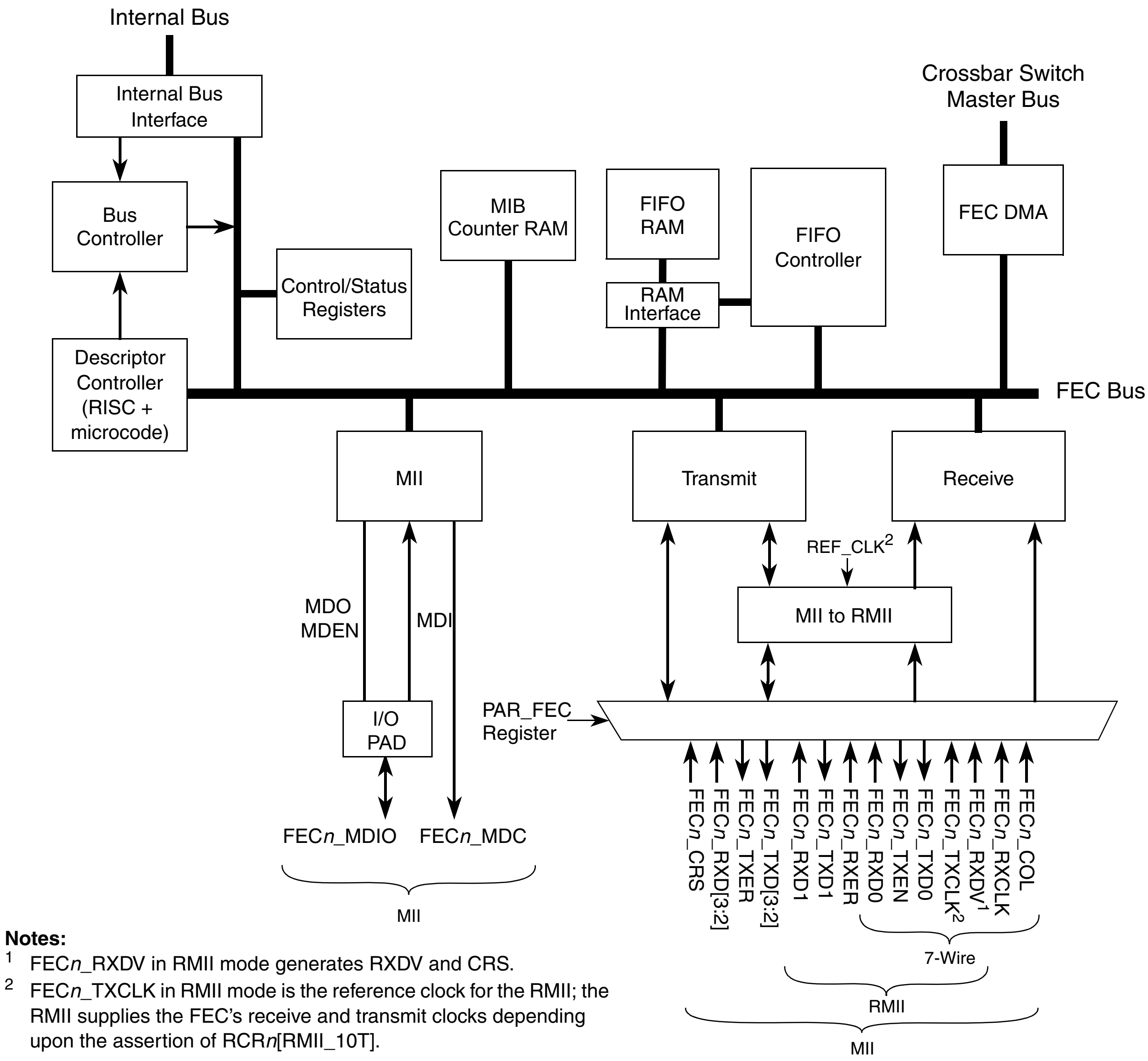
The Ethernet media access controller (MAC) supports 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FECs support five different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FECs support the 10/100 Mbps MII, 10/100 Mbps reduced MII, and the 10 Mbps-only 7-wire interface.

NOTE

The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, "Pin Multiplexing and Control") prior to configuring the FECs.

26.1.2 Block Diagram

Figure 26-1 shows the block diagram of a single FEC. The FECs are implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.



Notes:

[1] FEC*n*_RXDV in RMII mode generates RXDV and CRS.

[2] FEC*n*_TXCLK in RMII mode is the reference clock for the RMII; the RMII supplies the FEC's receive and transmit clocks depending upon the assertion of RCR*n*[RMII_10T].

Figure 26-1. FEC*n* Block Diagram

The descriptor controller is a RISC-based controller providing these functions in the FECs:

- Initialization (those internal registers not initialized by you or hardware)
- High level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

NOTE

DMA references in this section refer to the FEC's DMA engine. This DMA engine transfers FEC data only and is not related to the eDMA controller described in Chapter 19, "Enhanced Direct Memory Access (eDMA)," nor to the DMA timers described in Chapter 30, "DMA Timers (DTIM0–DTIM3)."

The RAM is the focal point of all data flow in the Fast Ethernet controller and divides into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR*n* register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block, and receive data flows from the receive block into the receive FIFO.

You control the FECs by writing into control registers located in each block. The CSR (control and status registers) block provides global control (Ethernet reset and enable) and interrupt managing registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the FEC*n*_MDC (management data clock) and FEC*n*_MDIO (management data input/output) lines of the MII interface.

The FEC DMA block (not to be confused with the device's eDMA controller) provides multiple channels allowing transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

The transmit and receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The message information block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC, but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters. See Section 26.4.1, "MIB Block Counters Memory Map," for more information.

26.1.3 Features

The FECs incorporate the following features:

- Support for five different Ethernet physical interfaces:
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 100-Mbps reduced media independent interface (RMII)
  - 10-Mbps reduced media independent interface (RMII)
  - 10-Mbps 7-wire interface (industry standard)
- IEEE 802.3 full duplex flow control
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- Support for full-duplex operation (200 Mbps throughput) with a minimum internal bus clock rate of 50 MHz
- Support for half-duplex operation (100 Mbps throughput) with a minimum internal bus clock rate of 50 MHz

- Retransmission from transmit FIFO following a collision (no processor bus utilization)
- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor bus utilization)
- Address recognition
  - — Frames with broadcast address may be always accepted or always rejected
  - — Exact match for single 48-bit individual (unicast) address
  - — Hash (64-bit hash) check of individual (unicast) addresses
  - — Hash (64-bit hash) check of group (multicast) addresses
  - — Promiscuous mode

26.2 Modes of Operation

The primary operational modes are described in this section.

26.2.1 Full and Half Duplex Operation

Full duplex mode is for use on point-to-point links between switches or end node to switch. Half duplex mode works in connections between an end node and a repeater or between repeaters. TCR*n*[FDEN] controls duplex mode selection.

When configured for full duplex mode, flow control may be enabled. Refer to the TCR*n*[RFC_PAUSE,TFC_PAUSE] bits, the RCR*n*[FCE] bit, and Section 26.5.11, "Full Duplex Flow Control," for more details.

26.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in Section 26.5.6, "Network Interface Options."

26.2.2.1 10 Mbps and 100 Mbps MII Interface

The IEEE 802.3 standard defines the media independent interface (MII) for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by setting RCR*n*[MII_MODE].

FEC*n*_TXCLK and FEC*n*_RXCLK pins driven by the external transceiver determine the operation speed. The transceiver auto-negotiates the speed or software controls it via the serial management interface (FEC*n*_MDC/FEC*n*_MDIO pins) to the transceiver. Refer to the MMFR*n* and MSCR*n* register descriptions, as well as the section on the MII, for a description of how to read and write registers in the transceiver via this interface.

26.2.2.2 10 Mbps and 100 Mbps RMII Interface

The reduced media independent interface (RMII) is a low cost alternative to the IEEE 802.3 MII standard. This interface provides the functionality of the MII interface on a total of 8 pins instead of 18. The RMII interface for 10/100 Ethernet MAC-PHY interface was defined by an industry consortium and is not currently included in the IEEE 802.3 standard. The PAR_FEC register in the pin multiplexing and control module controls this functionality which is reflected in the read-only RCR*n*[RMII_MODE] bit. The RCR*n*[RMII_10T] bit determines the speed of operation. The reference clock for RMII is always 50 MHz, but this clock can be divided by 10 within the RCR register to support 10 Mbps operation. The PHY must be configured accordingly. See Chapter 16, “Pin Multiplexing and Control,” for more details on the PAR_FEC register.

26.2.2.3 10 Mpbs 7-Wire Interface Operation

The FECs support 7-wire interface used by many 10 Mbps Ethernet transceivers. The RCR[MII_MODE] bit controls this functionality. If this bit is cleared, MII mode is disabled and the 10 Mbps 7-wire mode is enabled.

26.2.3 Address Recognition Options

The address options supported are promiscuous, broadcast reject, individual address (hash or exact match), and multicast hash match. Address recognition options are discussed in detail in Section 26.5.9, “Ethernet Address Recognition.”

26.2.4 Internal Loopback

Internal loopback mode is selected via RCR*n*[LOOP]. Loopback mode is discussed in detail in Section 26.5.14, “MII Internal and External Loopback.”

26.3 External Signal Description

Table 26-1 describes the various FEC signals, as well as indicating which signals work in available modes.

Table 26-1. FEC Signal Descriptions

Signal Name	MII	7-wire	RMII	Description
FEC_COL	X	X	—	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.
FEC_CRS	X	—	—	Carrier sense. When asserted, indicates transmit or receive medium is not idle. In RMII mode, this signal is present on the FEC_RXDV pin.
FEC_MDC	X	—	—	Output clock provides a timing reference to the PHY for data transfers on the FEC_MDIO signal.
FEC_MDIO	X	—	—	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC_MDC. This signal is an input after reset. When the FEC operates in 10Mbps 7-wire interface mode, this signal should be connected to VSS.
FEC_RXCLK	X	X	—	Provides a timing reference for FEC_RXDV, FEC_RXD[3:0], and FEC_RXER.
FEC_RXDV	X	X	X	Asserting the FEC_RXDV input indicates PHY has valid nibbles present on the MII. FEC_RXDV must remain asserted from the first recovered nibble of the frame through to the last nibble. Assertion of FEC_RXDV must start no later than the SFD and exclude any EOF. In RMII mode, this pin also generates the CRS signal.
FEC_RXD0	X	X	X	This pin contains the Ethernet input data transferred from PHY to the media-access controller when FEC_RXDV is asserted.

Table 26-1. FEC Signal Descriptions (continued)

Signal Name	MII	7-wire	RMII	Description
FEC_RXD1	X	—	X	This pin contains the Ethernet input data transferred from PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXD[3:2]	X	—	—	These pins contain the Ethernet input data transferred from PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXER	X	—	X	When asserted with FEC_RXDV, indicates PHY detects an error in the current frame. When FEC_RXDV is not asserted, FEC_RXER has no effect.
FEC_TXCLK	X	X	X	Input clock which provides a timing reference for FEC_TXEN, FEC_TXD[3:0] and FEC_TXER. In RMII mode, this signal is the reference clock for receive, transmit, and the control interface.
FEC_TXD0	X	X	X	The serial output Ethernet data and only valid during the assertion of FEC_TXEN.
FEC_TXD1	X	—	X	This pin contains the serial output Ethernet data and valid only during assertion of FEC_TXEN.
FEC_TXD[3:2]	X	—	—	These pins contain the serial output Ethernet data and valid only during assertion of FEC_TXEN.
FEC_TXEN	X	X	X	Indicates when valid nibbles are present on the MII. This signal is asserted with the first nibble of a preamble and is negated before the first FEC_TXCLK following the final nibble of the frame.
FEC_TXER	X	—	—	When asserted for one or more clock cycles while FEC_TXEN is also asserted, PHY sends one or more illegal symbols. FEC_TXER has no effect at 10 Mbps or when FEC_TXEN is negated.

26.4 Memory Map/Register Definition

The FECs are programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs control operation modes and extract global status information. The descriptors pass data buffers and related buffer information between the hardware and software.

Each FEC implementation requires a 1-Kbyte memory map space, which is divided into two sections of 512 bytes each for:

- Control/status registers
- Event/statistic counters held in the MIB block

Table 26-2 defines the top level memory map.

Table 26-2. Module Memory Map

Address	Function
0xFC03_0000 – FC03_01FF	FEC0 Control/Status Registers
0xFC03_0200 – FC03_02FF	FEC0 MIB Block Counters
0xFC03_4000 – FC03_41FF	FEC1 Control/Status Registers
0xFC03_4200 – FC03_42FF	FEC1 MIB Block Counters

Table 26-3 shows the FEC register memory map.

Table 26-3. FEC Register Memory Map

Address FEC0 FEC1	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_0004 0xFC03_4004	Interrupt Event Register (EIR*n*)	32	R/W	0x0000_0000	26.4.2/26-11
0xFC03_0008 0xFC03_4008	Interrupt Mask Register (EIMR*n*)	32	R/W	0x0000_0000	26.4.3/26-13
0xFC03_0010 0xFC03_4010	Receive Descriptor Active Register (RDAR*n*)	32	R/W	0x0000_0000	26.4.4/26-14
0xFC03_0014 0xFC03_4014	Transmit Descriptor Active Register (TDAR*n*)	32	R/W	0x0000_0000	26.4.5/26-14
0xFC03_0024 0xFC03_4024	Ethernet Control Register (ECR*n*)	32	R/W	0xF000_0000	26.4.6/26-15
0xFC03_0040 0xFC03_4040	MII Management Frame Register (MMFR*n*)	32	R/W	Undefined	26.4.7/26-16
0xFC03_0044 0xFC03_4044	MII Speed Control Register (MSCR*n*)	32	R/W	0x0000_0000	26.4.8/26-17
0xFC03_0064 0xFC03_4064	MIB Control/Status Register (MIBC*n*)	32	R/W	0x0000_0000	26.4.9/26-18
0xFC03_0084 0xFC03_4084	Receive Control Register (RCR*n*)	32	R/W	0x05EE_0001	26.4.10/26-19
0xFC03_00C4 0xFC03_40C4	Transmit Control Register (TCR*n*)	32	R/W	0x0000_0000	26.4.11/26-21
0xFC03_00E4 0xFC03_40E4	Physical Address Low Register (PALR*n*)	32	R/W	Undefined	26.4.12/26-22
0xFC03_00E8 0xFC03_40E8	Physical Address High Register (PAUR*n*)	32	R/W	See Section	26.4.13/26-22
0xFC03_00EC 0xFC03_40EC	Opcode/Pause Duration (OPD*n*)	32	R/W	See Section	26.4.14/26-23
0xFC03_0118 0xFC03_4118	Descriptor Individual Upper Address Register (IAUR*n*)	32	R/W	Undefined	26.4.15/26-23
0xFC03_011C 0xFC03_411C	Descriptor Individual Lower Address Register (IALR*n*)	32	R/W	Undefined	26.4.16/26-24
0xFC03_0120 0xFC03_4120	Descriptor Group Upper Address Register (GAUR*n*)	32	R/W	Undefined	26.4.17/26-24
0xFC03_0124 0xFC03_4124	Descriptor Group Lower Address Register (GALR*n*)	32	R/W	Undefined	26.4.18/26-25
0xFC03_0144 0xFC03_4144	Transmit FIFO Watermark (TFWR*n*)	32	R/W	0x0000_0000	26.4.19/26-25

Table 26-3. FEC Register Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
FEC0 FEC1					
0xFC03_014C 0xFC03_414C	FIFO Receive Bound Register (FRBR*n*)	32	R	0x0000_0600	26.4.20/26-26
0xFC03_0150 0xFC03_4150	FIFO Receive FIFO Start Register (FRSR*n*)	32	R	0x0000_0500	26.4.21/26-26
0xFC03_0180 0xFC03_4180	Pointer to Receive Descriptor Ring (ERDSR*n*)	32	R/W	Undefined	26.4.22/26-27
0xFC03_0184 0xFC03_4184	Pointer to Transmit Descriptor Ring (ETDSR*n*)	32	R/W	Undefined	26.4.23/26-27
0xFC03_0188 0xFC03_4188	Maximum Receive Buffer Size (EMRBR*n*)	32	R/W	Undefined	26.4.24/26-28

26.4.1 MIB Block Counters Memory Map

The MIB counters memory map (Table 26-4) defines the locations in the MIB RAM space where hardware-maintained counters reside. The counters are divided into two groups:

- RMON counters include the Ethernet statistics counters defined in RFC 1757
- A counter is included to count truncated frames since only frame lengths up to 2047 bytes are supported

The transmit and receive RMON counters are independent, which ensures accurate network statistics when operating in full duplex mode.

The included IEEE counters support the mandatory and recommended counter packages defined in Section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The FEC supports IEEE Basic Package objects, but these do not require counters in the MIB block. In addition, some of the recommended package objects supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are also included.

Table 26-4. MIB Counters Memory Map

Address	Register
FEC0 FEC1	
0xFC03_0200 0xFC03_4200	Count of frames not counted correctly (RMON_T_DROP*n*)
0xFC03_0204 0xFC03_4204	RMON Tx packet count (RMON_T_PACKETS*n*)
0xFC03_0208 0xFC03_4208	RMON Tx broadcast packets (RMON_T_BC_PKT*n*)

Table 26-4. MIB Counters Memory Map (continued)

Address	Register
FEC0 FEC1	
0xFC03_020C 0xFC03_420C	RMON Tx multicast packets (RMON_T_MC_PKT*n*)
0xFC03_0210 0xFC03_4210	RMON Tx packets with CRC/align error (RMON_T_CRC_ALIGN*n*)
0xFC03_0214 0xFC03_4214	RMON Tx packets < 64 bytes, good CRC (RMON_T_UNDERSIZE*n*)
0xFC03_0218 0xFC03_4218	RMON Tx packets > MAX_FL bytes, good CRC (RMON_T_OVERSIZE*n*)
0xFC03_021C 0xFC03_421C	RMON Tx packets < 64 bytes, bad CRC (RMON_T_FRAG*n*)
0xFC03_0220 0xFC03_4220	RMON Tx packets > MAX_FL bytes, bad CRC (RMON_T_JAB*n*)
0xFC03_0224 0xFC03_4224	RMON Tx collision count (RMON_T_COL*n*)
0xFC03_0228 0xFC03_4228	RMON Tx 64 byte packets (RMON_T_P64*n*)
0xFC03_022C 0xFC03_422C	RMON Tx 65 to 127 byte packets (RMON_T_P65TO127*n*)
0xFC03_0230 0xFC03_4230	RMON Tx 128 to 255 byte packets (RMON_T_P128TO255*n*)
0xFC03_0234 0xFC03_4234	RMON Tx 256 to 511 byte packets (RMON_T_P256TO511*n*)
0xFC03_0238 0xFC03_4238	RMON Tx 512 to 1023 byte packets (RMON_T_P512TO1023*n*)
0xFC03_023C 0xFC03_423C	RMON Tx 1024 to 2047 byte packets (RMON_T_P1024TO2047*n*)
0xFC03_0240 0xFC03_4240	RMON Tx packets with > 2048 bytes (RMON_T_P_GTE2048*n*)
0xFC03_0244 0xFC03_4244	RMON Tx Octets (RMON_T_OCTETS*n*)
0xFC03_0248 0xFC03_4248	Count of transmitted frames not counted correctly (IEEE_T_DROP*n*)
0xFC03_024C 0xFC03_424C	Frames transmitted OK (IEEE_T_FRAME_OK*n*)
0xFC03_0250 0xFC03_4250	Frames transmitted with single collision (IEEE_T_1COL*n*)
0xFC03_0254 0xFC03_4254	Frames transmitted with multiple collisions (IEEE_T_MCOL*n*)

Table 26-4. MIB Counters Memory Map (continued)

Address	Register
FEC0 FEC1	
0xFC03_0258 0xFC03_4258	Frames transmitted after deferral delay (IEEE_T_DEF*n*)
0xFC03_025C 0xFC03_425C	Frames transmitted with late collision (IEEE_T_LCOL*n*)
0xFC03_0260 0xFC03_4260	Frames transmitted with excessive collisions (IEEE_T_EXCOL*n*)
0xFC03_0264 0xFC03_4264	Frames transmitted with Tx FIFO underrun (IEEE_T_MACERR*n*)
0xFC03_0268 0xFC03_4268	Frames transmitted with carrier sense error (IEEE_T_CSERR*n*)
0xFC03_026C 0xFC03_426C	Frames transmitted with SQE error (IEEE_T_SQE*n*)
0xFC03_0270 0xFC03_4270	Flow control pause frames transmitted (IEEE_T_FDXFC*n*)
0xFC03_0274 0xFC03_4274	Octet count for frames transmitted without error (IEEE_T_OCTETS_OK*n*)
0xFC03_0280 0xFC03_4280	Count of received frames not counted correctly (RMON_R_DROP*n*)
0xFC03_0284 0xFC03_4284	RMON Rx packet count (RMON_R_PACKETS*n*)
0xFC03_0288 0xFC03_4288	RMON Rx broadcast packets (RMON_R_BC_PKT*n*)
0xFC03_028C 0xFC03_428C	RMON Rx multicast packets (RMON_R_MC_PKT*n*)
0xFC03_0290 0xFC03_4290	RMON Rx packets with CRC/Align error (RMON_R_CRC_ALIGN*n*)
0xFC03_0294 0xFC03_4294	RMON Rx packets < 64 bytes, good CRC (RMON_R_UNDERSIZE*n*)
0xFC03_0298 0xFC03_4298	RMON Rx packets > MAX_FL bytes, good CRC (RMON_R_OVERSIZE*n*)
0xFC03_029C 0xFC03_429C	RMON Rx packets < 64 bytes, bad CRC (RMON_R_FRAG*n*)
0xFC03_02A0 0xFC03_42A0	RMON Rx packets > MAX_FL bytes, bad CRC (RMON_R_JAB*n*)
0xFC03_02A4 0xFC03_42A4	Reserved (RMON_R_RESVD_0*n*)
0xFC03_02A8 0xFC03_42A8	RMON Rx 64 byte packets (RMON_R_P64*n*)

Table 26-4. MIB Counters Memory Map (continued)

Address FEC0 FEC1	Register
0xFC03_02AC 0xFC03_42AC	RMON Rx 65 to 127 byte packets (RMON_R_P65TO127*n*)
0xFC03_02B0 0xFC03_42B0	RMON Rx 128 to 255 byte packets (RMON_R_P128TO255*n*)
0xFC03_02B4 0xFC03_42B4	RMON Rx 256 to 511 byte packets (RMON_R_P256TO511*n*)
0xFC03_02B8 0xFC03_42B8	RMON Rx 512 to 1023 byte packets (RMON_R_P512TO1023*n*)
0xFC03_02BC 0xFC03_42BC	RMON Rx 1024 to 2047 byte packets (RMON_R_P1024TO2047*n*)
0xFC03_02C0 0xFC03_42C0	RMON Rx packets with > 2048 bytes (RMON_R_P_GTE2048*n*)
0xFC03_02C4 0xFC03_42C4	RMON Rx octets (RMON_R_OCTETS*n*)
0xFC03_02C8 0xFC03_42C8	Count of received frames not counted correctly (IEEE_R_DROP*n*)
0xFC03_02CC 0xFC03_42CC	Frames received OK (IEEE_R_FRAME_OK*n*)
0xFC03_02D0 0xFC03_42D0	Frames received with CRC error (IEEE_R_CRC*n*)
0xFC03_02D4 0xFC03_42D4	Frames received with alignment error (IEEE_R_ALIGN*n*)
0xFC03_02D8 0xFC03_42D8	Receive FIFO overflow count (IEEE_R_MACERR*n*)
0xFC03_02DC 0xFC03_42DC	Flow control pause frames received (IEEE_R_FDXFC*n*)
0xFC03_02E0 0xFC03_42E0	Octet count for frames received without error (IEEE_R_OCTETS_OK*n*)

26.4.2 Ethernet Interrupt Event Registers (EIR0 & EIR1)

When an event occurs that sets a bit in EIR*n*, an interrupt occurs if the corresponding bit in the interrupt mask register (EIMR) is also set. Writing a 1 to an EIR bit clears it; writing 0 has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts which may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC, and RL. Interrupts resulting from internal errors are HBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters:

- HBERR - IEEE_T_SQE
- BABR - RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT - RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)
- LATE_COL - IEEE_T_LCOL
- COL_RETRY_LIM - IEEE_T_EXCOL
- XFIFO_UN - IEEE_T_MACERR

Software may choose to mask off these interrupts because these errors are visible to network management via the MIB counters.

Address: 0xFC03_0004 (EIR0)
0xFC03_4004 (EIR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-2. Ethernet Interrupt Event Register (EIR*n*)

Table 26-5. EIR*n* Field Descriptions

Field	Description
31 HBERR	Heartbeat error. Indicates TCR*n*[HBC] is set and that the COL input was not asserted within the heartbeat window following a transmission.
30 BABR	Babbling receive error. Indicates a frame was received with length in excess of RCR*n*[MAX_FL] bytes.
29 BABT	Babbling transmit error. Indicates the transmitted frame length exceeds RCR*n*[MAX_FL] bytes. Usually this condition is caused by a frame that is too long is placed into the transmit data buffer(s). Truncation does not occur.
28 GRA	Graceful stop complete. Indicates the graceful stop is complete. During graceful stop the transmitter is placed into a pause state after completion of the frame currently being transmitted. This bit is set by one of three conditions: 1) A graceful stop initiated by the setting of the TCR*n*[GTS] bit is now complete. 2) A graceful stop initiated by the setting of the TCR*n*[TFC_PAUSE] bit is now complete. 3) A graceful stop initiated by the reception of a valid full duplex flow control pause frame is now complete. Refer to Section 26.5.11, "Full Duplex Flow Control."
27 TXF	Transmit frame interrupt. Indicates a frame has been transmitted and the last corresponding buffer descriptor has been updated.
26 TXB	Transmit buffer interrupt. Indicates a transmit buffer descriptor has been updated.
25 RXF	Receive frame interrupt. Indicates a frame has been received and the last corresponding buffer descriptor has been updated.

Table 26-5. EIR*n* Field Descriptions (continued)

Field	Description
24 RXB	Receive buffer interrupt. Indicates a receive buffer descriptor not the last in the frame has been updated.
23 MII	MII interrupt. Indicates the MII has completed the data transfer requested.
22 EBERR	Ethernet bus error. Indicates a system bus error occurred when a DMA transaction is underway. When the EBERR bit is set, ECR*n*[ETHER_EN] is cleared, halting frame processing by the FEC. When this occurs, software needs to ensure that the FIFO controller and DMA also soft reset.
21 LC	Late collision. Indicates a collision occurred beyond the collision window (slot time) in half duplex mode. The frame truncates with a bad CRC and the remainder of the frame is discarded.
20 RL	Collision retry limit. Indicates a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted and transmission of the next frame commences. This error can only occur in half duplex mode.
19 UN	Transmit FIFO underrun. Indicates the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.
18–0	Reserved, must be cleared.

26.4.3 Interrupt Mask Registers (EIMR0 & EIMR1)

The EIMR*n* registers control which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. A hardware reset clears this register. If the corresponding bits in the EIR*n* and EIMR*n* registers are set, an interrupt is generated. The interrupt signal remains asserted until a 1 is written to the EIR*n* bit (write 1 to clear) or a 0 is written to the EIMR*n* bit.

Address: 0xFC03_0008 (EIMR0)
0xFC03_4008 (EIMR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-3. Ethernet Interrupt Mask Register (EIMR*n*)

Table 26-6. EIMR*n* Field Descriptions

Field	Description
31–19 See Figure 26-3 and Table 26-5	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR*n* register. The corresponding EIMR*n* bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR*n* samples the signal generated by the interrupting source. The corresponding EIR*n* bit reflects the state of the interrupt signal even if the corresponding EIMR*n* bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
18–0	Reserved, must be cleared.

26.4.4 Receive Descriptor Active Registers (RDAR0 & RDAR1)

RDAR*n* is a command register, written by the user, indicating the receive descriptor ring is updated (the driver produced empty receive buffers with the empty bit set).

When the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and processes receive frames (provided ECR*n*[ETHER_EN] is also set). After the FEC polls a receive descriptor whose empty bit is not set, FEC clears the RDAR bit and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors are placed into the receive descriptor ring.

The RDAR registers are cleared at reset and when ECR*n*[ETHER_EN] is cleared.

Address: 0xFC03_0010 (RDAR0)
0xFC03_4010 (RDAR1)

Access: User 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
R	0	0	0	0	0	0	0	RDAR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-4. Receive Descriptor Active Register (RDAR*n*)

Table 26-7. RDAR*n* Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 RDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional empty descriptors remain in the receive ring. Also cleared when ECR*n*[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

26.4.5 Transmit Descriptor Active Registers (TDAR0 & TDAR1)

The TDAR*n* are command registers which the user writes to indicate the transmit descriptor ring is updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

When the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the FEC polls the transmit descriptor ring and processes transmit frames (provided ECR*n*[ETHER_EN] is also set). After the FEC polls a transmit descriptor that is a ready bit not set, FEC clears the TDAR bit and ceases transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors are placed into the transmit descriptor ring.

The TDAR*n* register is cleared at reset, when ECR*n*[ETHER_EN] is cleared, or when ECR*n*[RESET] is set.

Address: 0xFC03_0014 (TDAR0)
0xFC03_4014 (TDAR1)

Access: User 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
R	0	0	0	0	0	0	0	TDAR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. Transmit Descriptor Active Register (TDAR*n*)

Table 26-8. TDAR*n* Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 TDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional ready descriptors remain in the transmit ring. Also cleared when ECR*n*[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

26.4.6 Ethernet Control Registers (ECR0 & ECR1)

ECR*n* is a read/write user register, though hardware may alter fields in this register as well. The ECR*n* enables/disables the FEC.

Address: 0xFC03_0024 (ECR0)
0xFC03_4024 (ECR1)

Access: User 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
R	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ETHER_EN	RESET
W																																
Reset	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. Ethernet Control Register (ECR*n*)

Table 26-9. ECR*n* Field Descriptions

Field	Description
31–2	Reserved, must be cleared.

Table 26-9. ECR*n* Field Descriptions (continued)

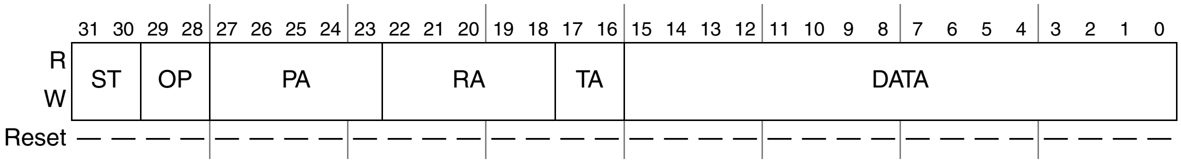
Field	Description
1 ETHER_EN	When this bit is set, FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception immediately stops and transmission stops after a bad CRC is appended to any currently transmitted frame. The buffer descriptor(s) for an aborted transmit frame are not updated after clearing this bit. When ETHER_EN is cleared, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. Hardware alters the ETHER_EN bit under the following conditions: • ECR*n*[RESET] is set by software, in which case ETHER_EN is cleared • An error condition causes the EIR*n*[EBERR] bit to set, in which case ETHER_EN is cleared
0 RESET	When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. ECR*n*[ETHER_EN] is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately eight internal bus clock cycles after this bit is set.

26.4.7 MII Management Frame Registers (MMFR0 & MMFR1)

The MMFR*n* is user-accessible and does not reset to a defined value. The MMFR*n* registers are used to communicate with the attached MII compatible PHY device(s), providing read/write access to their MII registers. Performing a write to the MMFR*n* causes a management frame to be sourced unless the MSCR*n* is programmed to 0. If MSCR*n* is cleared while MMFR*n* is written and then MSCR*n* is written with a non-zero value, an MII frame is generated with the data previously written to the MMFR*n*. This allows MMFR*n* and MSCR*n* to be programmed in either order if MSCR*n* is currently zero.

Address: 0xFC03_0040 (MMFR0)
0xFC03_4040 (MMFR1)

Access: User 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
R W	ST		OP		PA					RA					TA		DATA															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-7. MII Management Frame Register (MMFR*n*)

Table 26-10. MMFR*n* Field Descriptions

Field	Description
31–30 ST	Start of frame delimiter. These bits must be programmed to 0b01 for a valid MII management frame.
29–28 OP	Operation code. 00 Write frame operation, but not MII compliant. 01 Write frame operation for a valid MII management frame. 10 Read frame operation for a valid MII management frame. 11 Read frame operation, but not MII compliant.
27–23 PA	PHY address. This field specifies one of up to 32 attached PHY devices.
22–18 RA	Register address. This field specifies one of up to 32 registers within the specified PHY device.

Table 26-10. MMFR*n* Field Descriptions (continued)

Field	Description
17–16 TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
15–0 DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

To perform a read or write operation on the MII Management Interface, write the MMFR*n* register. To generate a valid read or write management frame, ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated, but does not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII Management Interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR*n* register. Writing this pattern causes the control logic to shift out the data in the MMFR*n* register following a preamble generated by the control state machine. During this time, contents of the MMFR*n* register are altered as the contents are serially shifted and are unpredictable if read by the user. After the write management frame operation completes, the MII interrupt is generated. At this time, contents of the MMFR*n* register match the original value written.

To generate an MII management interface read frame (read a PHY register), the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR*n* register (the content of the DATA field is a don’t care). Writing this pattern causes the control logic to shift out the data in the MMFR*n* register following a preamble generated by the control state machine. During this time, contents of the MMFR*n* register are altered as the contents are serially shifted and are unpredictable if read by the user. After the read management frame operation completes, the MII interrupt is generated. At this time, the contents of the MMFR*n* register match the original value written except for the DATA field whose contents are replaced by the value read from the PHY register.

If the MMFR*n* register is written while frame generation is in progress, the frame contents are altered. Software must use the MII interrupt to avoid writing to the MMFR*n* register while frame generation is in progress.

26.4.8 MII Speed Control Registers (MSCR0 & MSCR1)

The MSCR*n* provides control of the MII clock (FEC*n*_MDC pin) frequency and allows a preamble drop on the MII management frame.

Address: 0xFC03_0044 (MSCR0)
0xFC03_4044 (MSCR1)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DIS_PRE	MII_SPEED						0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-8. MII Speed Control Register (MSCRn)

Table 26-11. MSCR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 DIS_PRE	Setting this bit causes the preamble (32 ones) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY device(s) does not require it.
6–1 MII_SPEED	Controls the frequency of the MII management interface clock (FEC*n*_MDC) relative to the internal bus clock. A value of 0 in this field turns off the FEC*n*_MDC and leaves it in low voltage state. Any non-zero value results in the FEC*n*_MDC frequency of 1/(MII_SPEED × 2) of the internal bus frequency.
0	Reserved, must be cleared.

The MII_SPEED field must be programmed with a value to provide an FEC*n*_MDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII_SPEED must be set to a non-zero value to source a read or write management frame. After the management frame is complete, the MSCR*n* register may optionally be set to 0 to turn off the FEC*n*_MDC. The FEC*n*_MDC generated has a 50% duty cycle except when MII_SPEED changes during operation (change takes effect following a rising or falling edge of FEC*n*_MDC).

If the internal bus clock is 25 MHz, programming this register to 0x0000_0005 results in an FEC*n*_MDC as stated the equation below.

$$25\text{ MHz} \times \frac{1}{5 \times 2} = 2.5\text{ MHz} \qquad \textbf{\textit{Eqn. 26-1}}$$

A table showing optimum values for MII_SPEED as a function of internal bus clock frequency is provided below.

Table 26-12. Programming Examples for MSCR*n*

Internal FEC Clock Frequency	MSCR[MII_SPEED]	FEC*n*_MDC frequency
25 MHz	0x5	2.50 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.50 MHz
50 MHz	0xA	2.50 MHz
66 MHz	0xE	2.36 MHz

26.4.9 MIB Control Registers (MIBC0 & MIBC1)

The MIBC is a read/write register controlling and observing the state of the MIB block. User software accesses this register if there is a need to disable the MIB block operation. For example, to clear all MIB counters in RAM:

1. Disable the MIB block
2. Clear all the MIB RAM locations
3. Enable the MIB block

The MIB_DIS bit is reset to 1. See Table 26-4 for the locations of the MIB counters.

Address: 0xFC03_0064 (MIBC0)
0xFC03_4064 (MIBC1)

Access: User 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
R	MIB_DIS	MIB_IDLE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. MIB Control Register (MIBC*n*)

Table 26-13. MIBC*n* Field Descriptions

Field	Description
31 MIB_DIS	A read/write control bit. If set, the MIB logic halts and not update any MIB counters.
30 MIB_IDLE	A read-only status bit. If set the MIB block is not currently updating any MIB counters.
29–0	Reserved.

26.4.10 Receive Control Registers (RCR0 & RCR1)

RCR*n* controls the operational mode of the receive block and must be written only when ECR*n*[ETHER_EN] is cleared (initialization time).

Address: 0xFC03_0084 (RCR0)
0xFC03_4084 (RCR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	RMII_ECHO	RMII_LOOP	RMII_10T	RMII_MODE	0	0	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 26-10. Receive Control Register (RCR*n*)

Table 26-14. RCR*n* Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26–16 MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL causes the BABT interrupt to occur. Receive frames longer than MAX_FL causes the BABR interrupt to occur and sets the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed is 1518 or 1522 if VLAN tags are supported.

Table 26-14. RCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–12</td><td>Reserved, must be cleared.</td></tr>
<tr><td>11
RMII_ECHO</td><td>RMII Echo. Enables RMII echo mode. RMII 2-bit receive data is processed through the RMII receive logic to the FEC and also routes through the RMII’s transmit logic to become RMII 2-bit transmit data out.
0 Normal operation.
1 RMII echo mode enabled.
<b>Note:</b> When RMII_ECHO is set, proper operation is guaranteed only when RMII_MODE = 1, RMII_LOOP = 0 and LOOP = 0.</td></tr>
<tr><td>10
RMII_LOOP</td><td>RMII loopback. Enables RMII loopback mode. Causes the MII transmit outputs from the Ethernet controller to loop back to the Ethernet controllers’s MII receive inputs through the RMII transmit/receive logic.
0 Normal operation.
1 RMII loopback mode enabled.
<b>Note:</b> When RMII_LOOP is set, proper operation is guaranteed only when RMII_MODE = 1, RMII_ECHO = 0, LOOP = 0, and TCR<i>n</i>[FDEN] = 1.</td></tr>
<tr><td>9
RMII_10T</td><td>RMII 10-Base T. Enables 10Mbps mode of the RMII. Determines the clock frequency of the clock source to the FEC logic to support 10/100Mbps operations.
0 100 Mbps operation. The 50 MHz RMII reference clock on FEC_TXCLK is sent to the RMII, while a divided-by-2 version (25 MHz) is sent to the FEC.
1 10 Mbps operation. The 50 MHz RMII reference clock on FEC_TXCLK is divided by 10 (5 MHz) and sent to the RMII, while a divided-by-20 version (2.5 MHz) is sent to the FEC.</td></tr>
<tr><td>8
RMII_MODE</td><td>RMII Mode. Indicates if the FEC is in RMII or MII/7-wire mode. This is a read-only bit that reflects the status of the MII_MODE bit AND’d with the PAR_FEC register in the pin multiplexing and control module. See Section 16.3.5.1, “FEC Pin Assignment Register (PAR_FEC),” for more details.

For FEC0: RMII_MODE = RCR0[MII_MODE] && ($\overline{\text{PAR\_FEC[2]}}$ &amp;& PAR_FEC[1]) <b><i>Eqn. 26-2</i></b>

For FEC1: RMII_MODE = RCR1[MII_MODE] && ($\overline{\text{PAR\_FEC[6]}}$ &amp;& PAR_FEC[5]) <b><i>Eqn. 26-3</i></b>

0 FEC configured for MII or 7-wire mode as indicated by the MII_MODE bit.
1 FEC configured for RMII operation, only if the MII_MODE bit is set.

To summarize the various settings see the below table.
<table>
<tr><th>PAR_FEC[2:0]
PAR_FEC[6:5]</th><th>MII_MODE</th><th>RMII_MODE
(Equation 26-2)</th><th>Description</th></tr>
<tr><td>000</td><td><i>x</i></td><td>0</td><td>GPIO mode</td></tr>
<tr><td>001</td><td><i>x</i></td><td>0</td><td>Non-FEC functions</td></tr>
<tr><td rowspan="2">01<i>x</i></td><td>0</td><td>0</td><td>7-wire mode</td></tr>
<tr><td>1</td><td>1</td><td>RMII mode</td></tr>
<tr><td>10<i>x</i></td><td><i>x</i></td><td>0</td><td>Reserved</td></tr>
<tr><td>110</td><td><i>x</i></td><td>0</td><td>Reserved</td></tr>
<tr><td rowspan="2">111</td><td>0</td><td>0</td><td>7-wire mode</td></tr>
<tr><td>1</td><td>0</td><td>MII mode</td></tr>
</table>
</td></tr>
<tr><td>7–6</td><td>Reserved, must be cleared.</td></tr>
</table>

Table 26-14. RCR*n* Field Descriptions (continued)

Field	Description
5 FCE	Flow control enable. If asserted, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.
4 BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) equal to FFFF_FFFF_FFFF are rejected unless the PROM bit is set. If BC_REJ and PROM are set, frames with broadcast DA are accepted and the M (MISS) is set in the receive buffer descriptor.
3 PROM	Promiscuous mode. All frames are accepted regardless of address matching.
2 MII_MODE	Media independent interface mode. Selects the external interface mode for transmit and receive blocks. 0 7-wire mode (used only for serial 10 Mbps) 1 MII or RMII mode as indicated by the RMII_MODE bit
1 DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting (normally used for half duplex mode).
0 LOOP	Internal loopback. If set, transmitted frames are looped back internal to the device and transmit output signals are not asserted. The internal bus clock substitutes for the FEC*n*_TXCLK when LOOP is asserted. DRT must be set to 0 when setting LOOP.

26.4.11 Transmit Control Registers (TCR0 & TCR1)

TCR*n* is read/write and configures the transmit block. This register is cleared at system reset. Bits 2 and 1 must be modified only when ECR*n*[ETHER_EN] is cleared.

Address: 0xFC03_00C4 (TCR0)
0xFC03_40C4 (TCR1)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RFC_ PAUSE	TFC_ PAUSE	FDEN	HBC	GTS
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-11. Transmit Control Register (TCR*n*)

Table 26-15. TCR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 RFC_PAUSE	Receive frame control pause. This read-only status bit is asserted when a full duplex flow control pause frame is received and the transmitter pauses for the duration defined in this pause frame. This bit automatically clears when the pause duration is complete.
3 TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC stops transmission of data frames after the current transmission is complete. At this time, GRA interrupt in the EIR*n* register is asserted. With transmission of data frames stopped, MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFC_PAUSE bit and resumes transmitting data frames. If the transmitter pauses due to user assertion of GTS or reception of a PAUSE frame, the MAC may continue transmitting a MAC Control PAUSE frame.

Table 26-15. TCR Field Descriptions (continued)

Field	Description
2 FDEN	Full duplex enable. If set, frames transmit independent of carrier sense and collision inputs. This bit should only be modified when ECR*n*[ETHER_EN] is cleared.
1 HBC	Heartbeat control. If set, the heartbeat check performs following end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should only be modified when ECR*n*[ETHER_EN] is cleared.
0 GTS	Graceful transmit stop. When this bit is set, MAC stops transmission after any frame currently transmitted is complete and GRA interrupt in the EIR*n* register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. After transmission finishes, clear GTS to restart. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS is set, transmission stops after the collision. The frame is transmitted again after GTS is cleared. There may be old frames in the transmit FIFO that transmit when GTS is reasserted. To avoid this, clear ECR*n*[ETHER_EN] following the GRA interrupt.

26.4.12 Physical Address Lower Registers (PALR0 & PALR1)

PALR*n* contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the 6-byte source address field when transmitting PAUSE frames. This register is not reset and you must initialize it.

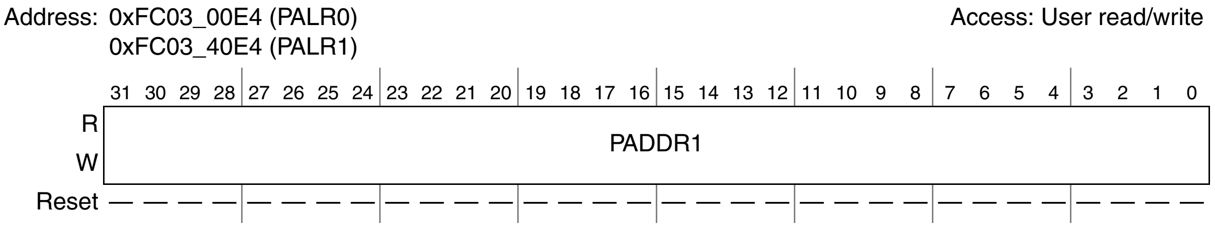


Figure 26-12. Physical Address Lower Register (PALR*n*)

Table 26-16. PALR*n* Field Descriptions

Field	Description
31–0 PADDR1	Bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8), and 3 (bits 7:0) of the 6-byte individual address are used for exact match and the source address field in PAUSE frames.

26.4.13 Physical Address Upper Registers (PAUR0 & PAUR1)

PAUR*n* contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 15:0 of PAUR*n* contain a constant type field (0x8808) for transmission of PAUSE frames. The upper 16 bits of this register are not reset and you must initialize it.

Address: 0xFC03_00E8 (PAUR0)
0xFC03_40E8 (PAUR1)

Access: User 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
R	PADDR2																TYPE															
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

Figure 26-13. Physical Address Upper Register (PAUR*n*)

Table 26-17. PAUR*n* Field Descriptions

Field	Description
31–16 PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address used for exact match, and the source address field in PAUSE frames.
15–0 TYPE	Type field in PAUSE frames. These 16 read-only bits are a constant value of 0x8808.

26.4.14 Opcode/Pause Duration Registers (OPD0 & OPD1)

The OPD*n* is read/write accessible. This register contains the 16-bit opcode and 16-bit pause duration fields used in transmission of a PAUSE frame. The opcode field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration field. The lower 16 bits of this register are not reset and you must initialize them.

Address: 0xFC03_00EC (OPD0)
0xFC03_40EC (OPD1)

Access: User 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
R	OPCODE																PAUSE_DUR															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-14. Opcode/Pause Duration Register (OPD*n*)

Table 26-18. OPD*n* Field Descriptions

Field	Description
31–16 OPCODE	Opcode field used in PAUSE frames. These read-only bits are a constant, 0x0001.
15–0 PAUSE_DUR	Pause Duration field used in PAUSE frames.

26.4.15 Descriptor Individual Upper Address Registers (IAUR0 & IAUR1)

IAUR*n* contains the upper 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the destination address (DA) field of receive frames with an individual DA. This register is not reset and you must initialize it.

Address: 0xFC03_0118 (IAUR0)
0xFC03_4118 (IAUR1)

Access: User read/write

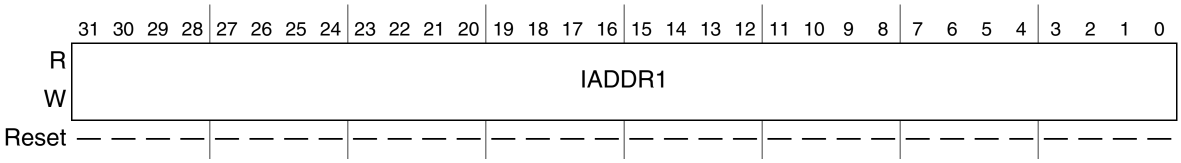


Figure 26-15. Descriptor Individual Upper Address Register (IAUR*n*)

Table 26-19. IAUR*n* Field Descriptions

Field	Description
31–0 IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

26.4.16 Descriptor Individual Lower Address Registers (IALR0 & IALR1)

IALR*n* contains the lower 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the DA field of receive frames with an individual DA. This register is not reset and you must initialize it.

Address: 0xFC03_011C (IALR0)
0xFC03_411C (IALR1)

Access: User read/write

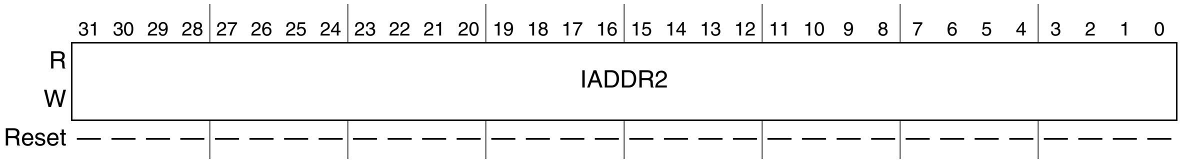


Figure 26-16. Descriptor Individual Lower Address Register (IALR*n*)

Table 26-20. IALR*n* Field Descriptions

Field	Description
31–0 IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0.

26.4.17 Descriptor Group Upper Address Registers (GAUR0 & GAUR1)

GAUR*n* contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

Address: 0xFC03_0120 (GAUR0)
0xFC03_4120 (GAUR1)

Access: User read/write

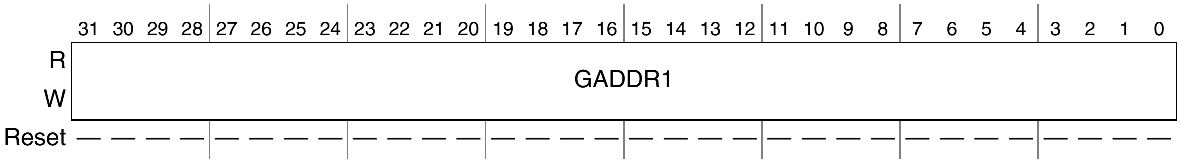


Figure 26-17. Descriptor Group Upper Address Register (GAUR*n*)

Table 26-21. GAUR*n* Field Descriptions

Field	Description
31–0 GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

26.4.18 Descriptor Group Lower Address Registers (GALR0 & GALR1)

GALR*n* contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

Address: 0xFC03_0124 (GALR0)
0xFC03_4124 (GALR1)

Access: User 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
R	GADDR2																															
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-18. Descriptor Group Lower Address Register (GALR*n*)

Table 26-22. GALR*n* Field Descriptions

Field	Description
31–0 GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

26.4.19 Transmit FIFO Watermark Registers (TFWR0 & TFWR1)

The TFWR*n* controls the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows you to minimize transmit latency (TFWR = 00 or 01) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

Address: 0xFC03_0144 (TFWR0)
0xFC03_4144 (TFWR1)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TFWR	
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-19. Transmit FIFO Watermark Register (TFWR*n*)

Table 26-23. TFWR*n* Field Descriptions

Field	Description
31–2	Reserved, must be cleared.
1–0 TFWR	Number of bytes written to transmit FIFO before transmission of a frame begins 00 64 bytes written 01 64 bytes written 10 128 bytes written 11 192 bytes written

26.4.20 FIFO Receive Bound Registers (FRBR0 & FRBR1)

FRBR*n* indicates the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR*n*, to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Address: 0xFC03_014C (FRBR0)
0xFC03_414C (FRBR1)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	R_BOUND								0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 26-20. FIFO Receive Bound Register (FRBR*n*)

Table 26-24. FRBR*n* Field Descriptions

Field	Description
31–10	Reserved, read as 0 (except bit 10, which is read as 1).
9–2 R_BOUND	Read-only. Highest valid FIFO RAM address.
1–0	Reserved, read as 0.

26.4.21 FIFO Receive Start Registers (FRSR0 & FRSR1)

FRSR*n* indicates the starting address of the receive FIFO. FRSR*n* marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR*n*. The receive FIFO uses addresses from FRSR*n* to FRBR*n* inclusive.

Hardware initializes the FRSR*n* register at reset. FRSR*n* only needs to be written to change the default value.

Address: 0xFC03_0150 (FRSR0)
0xFC03_4150 (FRSR1)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	R_FSTART								0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0

Figure 26-21. FIFO Receive Start Register (FRSR*n*)

Table 26-25. FRSR*n* Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10	Reserved, must be set.
9–2 R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs. For proper operation, ensure that R_FSTART is set to 0x48 or greater.
1–0	Reserved, must be cleared.

26.4.22 Receive Descriptor Ring Start Registers (ERDSR0 & ERDSR1)

ERDSR*n* points to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized prior to operation.

Address: 0xFC03_0180 (ERDSR0)
0xFC03_4180 (ERDSR1)

Access: User 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
R	R_DES_START																														0	0
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-22. Ethernet Receive Descriptor Ring Start Register (ERDSR*n*)

Table 26-26. ERDSR*n* Field Descriptions

Field	Description
31–2 R_DES_START	Pointer to start of receive buffer descriptor queue.
1–0	Reserved, must be cleared.

26.4.23 Transmit Buffer Descriptor Ring Start Registers (ETSDR0 & ETSDR1)

ETSDR*n* provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). You should write zeros to bits 1 and 0. Hardware ignores non-zero values in these two bit positions.

This register is undefined at reset and must be initialized prior to operation.

Address: 0xFC03_0184 (ETSDR0)
0xFC03_4184 (ETSDR1)

Access: User 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
R	X_DES_START																														0	0
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-23. Transmit Buffer Descriptor Ring Start Register (ETDSR*n*)

Table 26-27. ETDSR*n* Field Descriptions

Field	Description
31–2 X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, must be cleared.

26.4.24 Receive Buffer Size Registers (EMRBR0 & EMRBR1)

The EMRBR*n* is a user-programmable register that dictates the maximum size of all receive buffers. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR*n* must be set to RCR*n*[MAX_FL] or larger. To properly align the buffer, EMRBR*n* must be evenly divisible by 16. To ensure this, bits 3–0 are forced low.

To minimize bus utilization (descriptor fetches), it is recommended that EMRBR*n* be greater than or equal to 256 bytes.

The EMRBR*n* register is undefined at reset and must be initialized by the user.

Address: 0xFC03_0188 (EMRBR0)
0xFC03_4188 (EMRBR1)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	R_BUF_SIZE							0	0	0	0
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 26-24. Receive Buffer Size Register (EMRBR*n*)

Table 26-28. EMRBR*n* Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10–4 R_BUF_SIZE	Maximum size of receive buffer size in bytes. To minimize bus utilization (descriptor fetches), set this field to 256 bytes (0x10) or larger. 0x10 256 + 15 bytes (minimum size recommended) 0x11 272 + 15 bytes ... 0x7F 2032 + 15 bytes. The FEC writes up to 2047 bytes in the receive buffer. If data larger than 2047 is received, the FEC truncates it and shows 0x7FF in the receive descriptor
3–0	Reserved, must be cleared.

26.5 Functional Description

This section describes the operation of the FECs, beginning with the buffer descriptors, the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FECs.

26.5.1 Buffer Descriptors

This section provides a description of the operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields.

26.5.1.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames resides in one or more memory buffers external to the FEC. Associated with each buffer is a buffer descriptor (BD), which contains a starting address (32-bit aligned pointer), data length, and status/control information (which contains the current state for the buffer). To permit maximum user flexibility, the BDs are also located in external memory and are read by the FEC DMA engine.

Software produces buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD*n*[E] or TxBD*n*[R] bit produces the buffer. Software writing to TDAR*n* or RDAR*n* tells the FEC that a buffer is placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and consumes the buffers after they have been produced. After the data DMA is complete and the DMA engine writes the buffer descriptor status bits, hardware clears RxBD*n*[E] or TxBD*n*[R] to signal the buffer has been consumed. Software may poll the BDs to detect when the buffers are consumed or may rely on the buffer/frame interrupts. The driver may process these buffers, and they can return to the free list.

The ECR*n*[ETHER_EN] bit operates as a reset to the BD/DMA logic. When ECR*n*[ETHER_EN] is cleared, the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before ECR*n*[ETHER_EN] is set.

The buffer descriptors operate as two separate rings. ERDSR*n* defines the starting address for receive BDs and ETDSR*n* defines the starting address for transmit BDs. The wrap (W) bit defines the last buffer descriptor in each ring. When W is set, the next descriptor in the ring is at the location pointed to by ERDSR*n* and ETDSR*n* for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

26.5.1.1.1 Driver/DMA Operation with Transmit BDs

Typically, a transmit frame is divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a second buffer, IP header in a third buffer, and Ethernet/IEEE 802.3 header in a fouth buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type field(s)), so the driver must provide this in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. TxBD*n*[TC], which must be set by the driver, determines whether the MAC or driver appends the CRC.

The driver (TxBD software producer) should set up Tx BDs so a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length, and control (W, L, TC, ABC) and then the TxBD*n*[R] bit should be set in reverse order (third, second, then first BD) to ensure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA controller could DMA the first BD before the second was made available, potentially causing a transmit FIFO underrun.

In the FEC, the driver notifies the DMA that new transmit frame(s) are available by writing to TDAR*n*. When this register is written to (data value is not significant) the FEC, RISC tells the DMA to read the next transmit BD in the ring. After started, the RISC + DMA continues to read and interpret transmit BDs in order and DMA the associated buffers until a transmit BD is encountered with the R bit cleared. At this point, the FEC polls this BD one more time. If the R bit is cleared the second time, RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to TDAR*n*.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

26.5.1.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore, the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR*n* register.

The driver (RxBD software producer) should set up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the L bit (1 indicates last buffer in frame), the frame status bits (if L is set), and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD is written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end-of-frame buffers, the receive BD is written with L set and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer is written with the length of the entire frame, not only the length of the last buffer.

For simplicity, the driver may assign a large enough default receive buffer length to contain an entire frame, keeping in mind that a malfunction on the network or out-of-spec implementation could result in giant frames. Frames of 2K (2048) bytes or larger are truncated by the FEC at 2047 bytes so software never sees a receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR*n* register. As frames are received, the FEC fills receive buffers and updates the associated BDs, then reads the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit cleared, it polls this BD once more. If RxBD*n*[E] is clear a second time, FEC stops reading receive BDs until the driver writes to RDAR*n*.

26.5.1.2 Ethernet Receive Buffer Descriptors (RxBD0 & RxBD1)

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in the second longword. When the buffer has been DMA’d, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first longword. The M, BC, MC, LG, NO, CR, OV, and TR bits in the first longword of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2	Data Length															
Offset + 4	Rx Data Buffer Pointer - A[31:16]															
Offset + 6	Rx Data Buffer Pointer - A[15:0]															

Figure 26-25. Receive Buffer Descriptor (RxBD*n*)

Table 26-29. Receive Buffer Descriptor Field Definitions

Word	Field	Description
Offset + 0	15 E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD is filled with received data, or data reception has aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	14 RO1	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
Offset + 0	13 W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ERDSR.*n*
Offset + 0	12 RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
Offset + 0	11 L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.

Table 26-29. Receive Buffer Descriptor Field Definitions (continued)

Word	Field	Description
Offset + 0	10–9	Reserved, must be cleared.
Offset + 0	8 M	Miss. Written by the FEC. This bit is set by the FEC for frames accepted in promiscuous mode, but flagged as a miss by the internal address recognition. Therefore, while in promiscuous mode, you can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
Offset + 0	7 BC	Set if the DA is broadcast (FFFF_FFFF_FFFF).
Offset + 0	6 MC	Set if the DA is multicast and not BC.
Offset + 0	5 LG	Rx frame length violation. Written by the FEC. A frame length greater than RCR*n*[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	4 NO	Receive non-octet aligned frame. Written by the FEC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L-bit is set. If this bit is set, the CR bit is not set.
Offset + 0	3	Reserved, must be cleared.
Offset + 0	2 CR	Receive CRC error. Written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L-bit is set.
Offset + 0	1 OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and are zero. This bit is valid only if the L-bit is set.
Offset + 0	0 TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set, the frame must be discarded and the other error bits must be ignored as they may be incorrect.
Offset + 2	15–0 Data Length	Data length. Written by the FEC. Data length is the number of octets written by the FEC into this BD's data buffer if L equals 0 (the value is equal to EMRBR), or the length of the frame including CRC if L is set. It is written by the FEC once as the BD is closed.
0ffset + 4	15–0 A[31:16]	RX data buffer pointer, bits [31:16][1]
Offset + 6	15–0 A[15:0]	RX data buffer pointer, bits [15:0]

[1] The receive buffer pointer, containing the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. The Ethernet controller never modifies this value.

NOTE

When the software driver sets an E bit in one or more receive descriptors, the driver should follow with a write to RDAR.*n*

26.5.1.3 Ethernet Transmit Buffer Descriptors (TxBD0 & TxBD1)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel's TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (TxBD*n*[R]) when DMA of the buffer is complete. In the TxBD, the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword and the buffer pointer in the second longword.

The FEC clears the R bit when the buffer is transferred. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See Section 26.4.1, “MIB Block Counters Memory Map,” for more details.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	R	TO1	W	TO2	L	TC	ABC	—	—	—	—	—	—	—	—	—
Offset + 2	Data Length															
Offset + 4	Tx Data Buffer Pointer - A[31:16]															
Offset + 6	Tx Data Buffer Pointer - A[15:0]															

Figure 26-26. Transmit Buffer Descriptor (TxBD*n*)

Table 26-30. Transmit Buffer Descriptor Field Definitions

Word	Field	Description
Offset + 0	15 R	Ready. Written by the FEC and you. 0 The data buffer associated with this BD is not ready for transmission. You are free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, prepared for transmission by you, has not been transmitted or currently transmits. You may write no fields of this BD after this bit is set.
Offset + 0	14 TO1	Transmit software ownership. This field is reserved for software use. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	13 W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ETDSR.*n*
Offset + 0	12 TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	11 L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame 1 The buffer is the last in the transmit frame
Offset + 0	10 TC	Transmit CRC. Written by user (only valid if L is set). 0 End transmission immediately after the last data byte 1 Transmit the CRC sequence after the last data byte
Offset + 0	9 ABC	Append bad CRC. Written by user (only valid if L is set). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value)
Offset + 0	8–0	Reserved, must be cleared.
Offset + 2	15–0 Data Length	Data length, written by user. Data length is the number of octets the FEC should transmit from this BD’s data buffer. It is never modified by the FEC.

Table 26-30. Transmit Buffer Descriptor Field Definitions (continued)

Word	Field	Description
Offset + 4	15–0 A[31:16]	Tx data buffer pointer, bits [31:16][1]
Offset + 6	15–0 A[15:0]	Tx data buffer pointer, bits [15:0]

[1] The transmit buffer pointer, containing the address of the associated data buffer, must always be evenly divisible by 4. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

After the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame is setting the R bit in the first BD for the frame. The driver must follow that with a write to TDAR*n* that triggers the FEC to poll the next BD in the ring.

26.5.2 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations you must initialize prior to enabling the FECs.

26.5.2.1 Hardware Controlled Initialization

In the FEC, hardware resets registers and control logic that generate interrupts. A hardware reset negates output signals and resets general configuration bits.

Other registers reset when the ECR*n*[ETHER_EN] bit is cleared (which is accomplished by a hard reset or software to halt operation). By clearing ECR*n*[ETHER_EN], configuration control registers such as the TCR*n* and RCR*n* are not reset, but the entire data path is reset.

Table 26-31. ECR*n*[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR*n*	Cleared
TDAR*n*	Cleared
Descriptor Controller block	Halt operation

26.5.3 User Initialization (Prior to Setting ECR*n*[ETHER_EN])

You need to initialize portions the FEC prior to setting the ECR*n*[ETHER_EN] bit. The exact values depend on the particular application. The sequence is not important.

Table 26-32 defines Ethernet MAC registers requiring initialization.

Table 26-32. User Initialization (Before ECR*n*[ETHER_EN])

Description
Initialize EIMR*n*
Clear EIR*n* (write 0xFFFF_FFFF)
TFWR*n* (optional)
IALR*n* / IAUR*n*
GAUR*n* / GALR*n*
PALR*n* / PAUR*n* (only needed for full duplex flow control)
OPD*n* (only needed for full duplex flow control)
RCR*n*
TCR*n*
MSCR*n* (optional)
Clear MIB_RAM*n*

Table 26-33 defines FEC FIFO/DMA registers that require initialization.

Table 26-33. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR*n* (optional)
Initialize EMRBR*n*
Initialize ERDSR*n*
Initialize ETDSR*n*
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

26.5.4 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR*n*[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, hardware is ready for operation.

Table 26-34 shows microcontroller initialization operations.

Table 26-34. Microcontroller Initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO

Table 26-34. Microcontroller Initialization (continued)

Description
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

26.5.5 User Initialization (After Setting ECR*n*[ETHER_EN])

After setting ECR*n*[ETHER_EN], you can set up the buffer/frame descriptors and write to TDAR*n* and RDAR*n*. Refer to Section 26.5.1, “Buffer Descriptors,” for more details.

26.5.6 Network Interface Options

The FECs support an MII and reduced MII interface for 10/100 Mbps Ethernet, as well as a 7-wire serial interface for 10 Mbps Ethernet. The RCR*n*[MII_MODE] and RCR*n*[RMII_MODE] bits selects the interface mode. In MII mode (RCR*n*[MII_MODE] set and RCR*n*[RMII_MODE] cleared), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. Table 26-35 shows these signals.

Table 26-35. MII Mode

Signal Description	EMAC pin
Transmit Clock	FEC*n*_TXCLK
Transmit Enable	FEC*n*_TXEN
Transmit Data	FEC*n*_TXD[3:0]
Transmit Error	FEC*n*_TXER
Collision	FEC*n*_COL
Carrier Sense	FEC*n*_CRS
Receive Clock	FEC*n*_RXCLK
Receive Data Valid	FEC*n*_RXDV
Receive Data	FEC*n*_RXD[3:0]
Receive Error	FEC*n*_RXER
Management Data Clock	FEC*n*_MDC
Management Data Input/Output	FEC*n*_MDIO

In RMII mode (RCR*n*[MII_MODE] set and RCR*n*[RMII_MODE] cleared), the EMAC supports 8 external signals. These signals are shown in Table 26-36 below.

Table 26-36. RMII Mode Configuration

Signal Description	EMAC Pin
Reference Clock	FEC*n*_TXCLK
Transmit Enable	FEC*n*_TXEN
Transmit Data	FEC*n*_TXD[1:0]
Receive Data Valid	FEC*n*_RXDV
Receive Data	FEC*n*_RXD[1:0]
Receive Error	FEC*n*_RXER

The 7-wire serial mode interface (RCR*n*[MII_MODE] cleared and RCR*n*[RMII_MODE] cleared or set) is generally referred to as AMD mode. Table 26-37 shows the 7-wire mode connections to the external transceiver.

Table 26-37. 7-Wire Mode Configuration

Signal description	EMAC Pin
Transmit Clock	FEC*n*_TXCLK
Transmit Enable	FEC*n*_TXEN
Transmit Data	FEC*n*_TXD[0]
Collision	FEC*n*_COL
Receive Clock	FEC*n*_RXCLK
Receive Data Valid	FEC*n*_RXDV
Receive Data	FEC*n*_RXD[0]

26.5.7 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. After ECR*n*[ETHER_EN] is set and data appears in the transmit FIFO, the Ethernet MAC can transmit onto the network. The Ethernet controller transmits bytes least significant bit (lsb) first.

When the transmit FIFO fills to the watermark (defined by TFWR*n*), MAC transmit logic asserts FEC*n*_TXEN and starts transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (FEC*n*_CRS is asserted). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See Section 26.5.17.1, “Transmission Errors,” for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data is transmitted, FCS (frame check sequence) or 32-bit cyclic redundancy check (CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Transmit logic automatically pads short frames (if the TC bit in the transmit buffer descriptor for the end of frame buffer is set).

Settings in the EIMR*n* determine interrupts generated to the buffer (TXB) and frame (TFINT).

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL bytes, BABT interrupt is asserted. However, the entire frame is transmitted (no truncation).

To pause transmission, set TCR*n*[GTS] (graceful transmit stop). The FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame finishes or terminates with a collision. After the transmitter has stopped, the GRA (graceful stop complete) interrupt is asserted. If TCR*n*[GTS] is cleared, the FEC resumes transmission with the next frame.

26.5.7.1 Duplicate Frame Transmission

The FEC fetches transmit buffer descriptors (TxBDs) and the corresponding transmit data continuously until the transmit FIFO is full. It does not determine whether the TxBD to be fetched is already being processed internally (as a result of a wrap). As the FEC nears the end of the transmission of one frame, it begins to DMA the data for the next frame. To remain one BD ahead of the DMA, it also fetches the TxBD for the next frame. It is possible that the FEC fetches from memory a BD that has already been processed but not yet written back (it is read a second time with the R bit remains set). In this case, the data is fetched and transmitted again.

Using at least three TxBDs fixes this problem for large frames, but not for small frames. To ensure correct operation for large or small frames, one of the following must be true:

- The FEC software driver ensures that there is always at least one TxBD with the ready bit cleared.
- Every frame uses more than one TxBD and every TxBD but the last is written back immediately after the data is fetched.
- The FEC software driver ensures a minimum frame size, *n*. The minimum number of TxBDs is then (Tx FIFO Size ÷ (n + 4)) rounded up to the nearest integer (though the result cannot be less than three). The default Tx FIFO size is 192 bytes; this size is programmable.

26.5.8 FEC Frame Reception

The FEC receivers work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking. The Ethernet controller receives serial data lsb first.

When the driver enables the FEC receiver by setting ECR*n*[ETHER_EN], it immediately starts processing receive frames. When FEC*n*_RXDV is asserted, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the receiver processes the frame. If a valid PA/SFD is not found, the frame is ignored.

In serial mode, the first 16 bit times of RX_D0 following assertion of FEC*n*_RXDV are ignored. Following the first 16 bit times, the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame are received, the FEC performs address recognition on the frame.

After a collision window (64 bytes) of data is received and if address recognition has not rejected the frame, the receive FIFO signals the frame is accepted and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to reject the frame. Therefore, no collision fragments are presented to you except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and after the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV, and TR status bits, and the frame length. See Section 26.5.17.2, “Reception Errors,” for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR*n* register. A receive error interrupt is a babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt occurs and the LG bit in the receive buffer descriptor (RxBD*n*) is set. See Section 26.5.1.2, “Ethernet Receive Buffer Descriptors (RxBD0 & RxBD1),” for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD*n*, writes the other frame status bits into the RxBD*n*, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR*n*, maskable by RFIEN bit in EIMR*n*), indicating that a frame is received and is in memory. The Ethernet controller then waits for a new frame.

26.5.9 Ethernet Address Recognition

The FECs filter the received frames based on destination address (DA) type — individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames appears in the figures below.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in Figure 26-27 illustrates the address recognition decisions made by the receive block, while Figure 26-28 illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR*n*[BC_REJ]) is cleared, then the frame is accepted unconditionally, as shown in Figure 26-27. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in Figure 26-28.

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller performs a group hash table lookup using the 64-entry hash table programmed in GAUR*n* and GALR*n*. If a hash match occurs, the receiver accepts the frame.

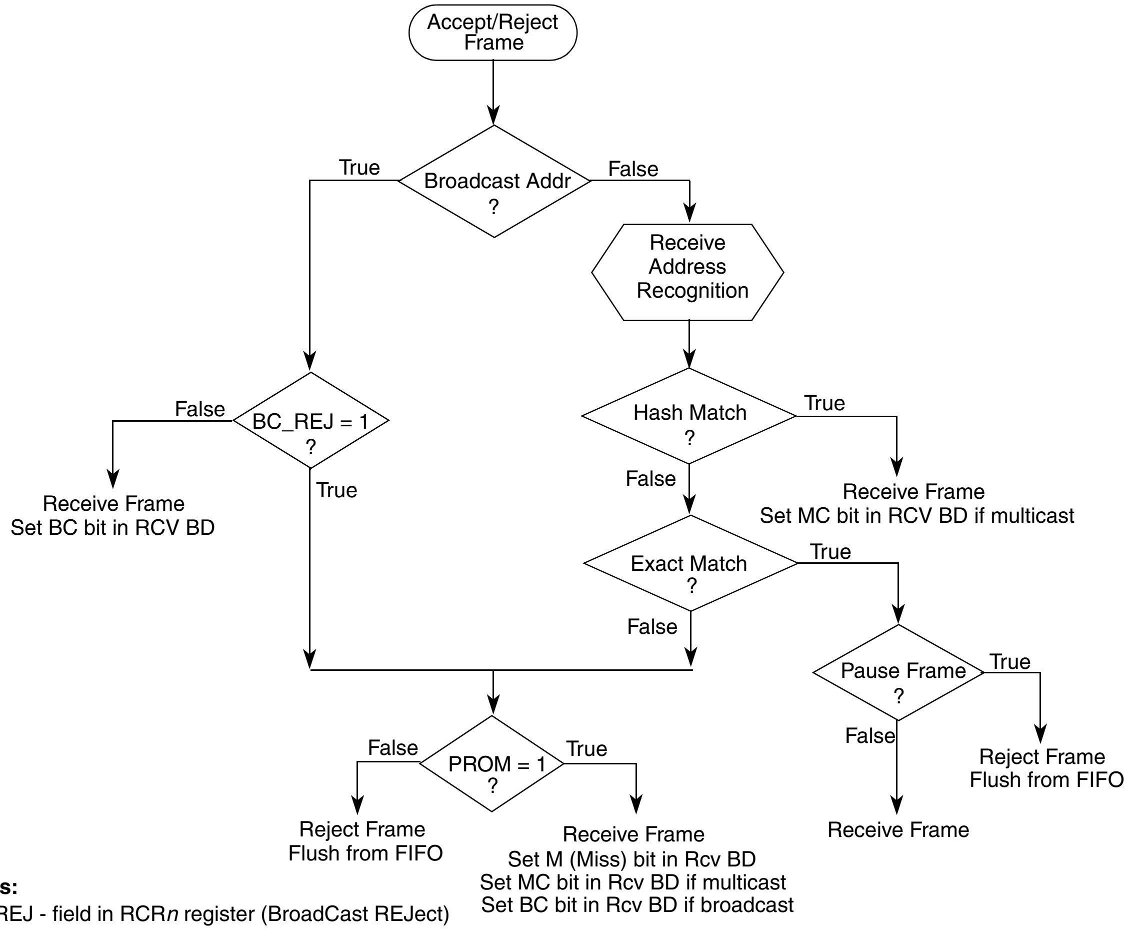
If flow control is enabled, the microcontroller does an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines the received frame is a valid PAUSE frame, the frame is rejected. The receiver detects a PAUSE frame with the DA field set to the designated PAUSE DA or the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that you program in the PALR*n* and PAUR*n* registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR*n* and IALR*n*. In the case of an individual hash match, the frame is accepted. Again, the receiver accepts or rejects the frame based on PAUSE frame detection, shown in Figure 26-27.

If neither a hash match (group or individual) nor an exact match (group or individual) occur, and if promiscuous mode is enabled (RCR*n*[PROM] set), the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

Similarly, if the DA is a broadcast address, broadcast reject (RCR*n*[BC_REJ]) is asserted, and promiscuous mode is enabled, the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

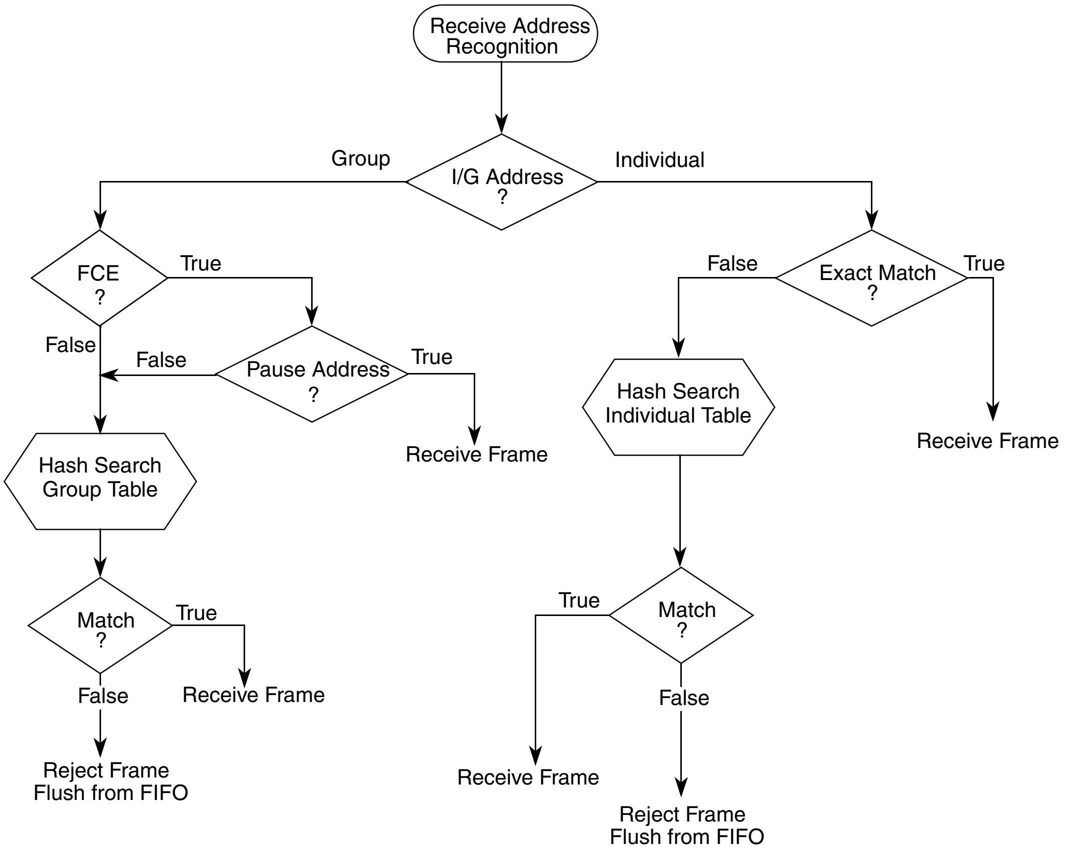
In general, when a frame is rejected, it is flushed from the FIFO.



Notes:

BC_REJ - field in RCR*n* register (BroadCast REJect)
PROM - field in RCR*n* register (PROMiscous mode)
Pause Frame - valid PAUSE frame received

Figure 26-27. Ethernet Address Recognition—Receive Block Decisions



Notes:
FCE - field in RCR*n* register (flow control enable)
I/G - Individual/Group bit in destination address (lsb in first byte received in MAC frame)

Figure 26-28. Ethernet Address Recognition—Microcode Decisions

26.5.10 Hash Algorithm

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, represented by 64 bits stored in GAUR*n*, GALR*n* (group address hash match), or IAUR*n*, IALR*n* (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the six most significant bits of the CRC-encoded result to generate a number between 0 and 63. The msb of the CRC result selects GAUR*n* (msb = 1) or GALR*n* (msb = 0). The five least significant bits of the hash result select the bit within the selected register. If the CRC generator selects a bit set in the hash table, the frame is accepted; otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The user must initialize the hash table registers. Use this CRC32 polynomial to compute the hash:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1 \qquad \textbf{\textit{Eqn. 26-4}}$$

Table 26-38 contains example destination addresses and corresponding hash values.

Table 26-38. Destination Address to 6-Bit Hash

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65FF_FFFF_FFFF	0x0	0
55FF_FFFF_FFFF	0x1	1
15FF_FFFF_FFFF	0x2	2
35FF_FFFF_FFFF	0x3	3
B5FF_FFFF_FFFF	0x4	4
95FF_FFFF_FFFF	0x5	5
D5FF_FFFF_FFFF	0x6	6
F5FF_FFFF_FFFF	0x7	7
DBFF_FFFF_FFFF	0x8	8
FBFF_FFFF_FFFF	0x9	9
BBFF_FFFF_FFFF	0xA	10
8BFF_FFFF_FFFF	0xB	11
0BFF_FFFF_FFFF	0xC	12
3BFF_FFFF_FFFF	0xD	13
7BFF_FFFF_FFFF	0xE	14
5BFF_FFFF_FFFF	0xF	15
27FF_FFFF_FFFF	0x10	16
07FF_FFFF_FFFF	0x11	17
57FF_FFFF_FFFF	0x12	18
77FF_FFFF_FFFF	0x13	19
F7FF_FFFF_FFFF	0x14	20
C7FF_FFFF_FFFF	0x15	21
97FF_FFFF_FFFF	0x16	22
A7FF_FFFF_FFFF	0x17	23
99FF_FFFF_FFFF	0x18	24
B9FF_FFFF_FFFF	0x19	25
F9FF_FFFF_FFFF	0x1A	26
C9FF_FFFF_FFFF	0x1B	27

Table 26-38. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
59FF_FFFF_FFFF	0x1C	28
79FF_FFFF_FFFF	0x1D	29
29FF_FFFF_FFFF	0x1E	30
19FF_FFFF_FFFF	0x1F	31
D1FF_FFFF_FFFF	0x20	32
F1FF_FFFF_FFFF	0x21	33
B1FF_FFFF_FFFF	0x22	34
91FF_FFFF_FFFF	0x23	35
11FF_FFFF_FFFF	0x24	36
31FF_FFFF_FFFF	0x25	37
71FF_FFFF_FFFF	0x26	38
51FF_FFFF_FFFF	0x27	39
7FFF_FFFF_FFFF	0x28	40
4FFF_FFFF_FFFF	0x29	41
1FFF_FFFF_FFFF	0x2A	42
3FFF_FFFF_FFFF	0x2B	43
BFFF_FFFF_FFFF	0x2C	44
9FFF_FFFF_FFFF	0x2D	45
DFFF_FFFF_FFFF	0x2E	46
EFFF_FFFF_FFFF	0x2F	47
93FF_FFFF_FFFF	0x30	48
B3FF_FFFF_FFFF	0x31	49
F3FF_FFFF_FFFF	0x32	50
D3FF_FFFF_FFFF	0x33	51
53FF_FFFF_FFFF	0x34	52
73FF_FFFF_FFFF	0x35	53
23FF_FFFF_FFFF	0x36	54
13FF_FFFF_FFFF	0x37	55
3DFF_FFFF_FFFF	0x38	56
0DFF_FFFF_FFFF	0x39	57
5DFF_FFFF_FFFF	0x3A	58
7DFF_FFFF_FFFF	0x3B	59

Table 26-38. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
FDFF_FFFF_FFFF	0x3C	60
DDFF_FFFF_FFFF	0x3D	61
9DFF_FFFF_FFFF	0x3E	62
BDFF_FFFF_FFFF	0x3F	63

26.5.11 Full Duplex Flow Control

Full-duplex flow control allows you to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable PAUSE frame detection, the FEC must operate in full-duplex mode (TCR*n*[FDEN] set) with flow control (RCR*n*[FCE] set). The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in Table 26-39. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 26-39. PAUSE Frame Field Specification

48-bit Destination Address	0x0180_C200_0001 or Physical Address
48-bit Source Address	Any
16-bit Type	0x8808
16-bit Opcode	0x0001
16-bit PAUSE Duration	0x0000 – 0xFFFF

The receiver and microcontroller modules perform PAUSE frame detection. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, TCR*n*[GTS] is set by the FEC internally. When transmission has paused, the EIR*n*[GRA] interrupt is asserted and the pause timer begins to increment. The pause timer uses the transmit backoff timer hardware for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD*n*[PAUSE_DUR] slot times have expired. On OPD*n*[PAUSE_DUR] expiration, TCR*n*[GTS] is cleared allowing MAC data frame transmission to resume. The receive flow control pause status bit (TCR*n*[RFC_PAUSE]) is set while the transmitter pauses due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and you must set flow control pause (TCR*n*[TFC_PAUSE]). After TCR*n*[TFC_PAUSE] is set, the transmitter sets TCR*n*[GTS] internally. When the transmission of data frames stops, the EIR*n*[GRA] (graceful stop complete) interrupt asserts and the pause frame is transmitted. TCR*n*[TFC_PAUSE,GTS] are then cleared internally.

You must specify the desired pause duration in the OPD*n* register.

When the transmitter pauses due to receiver/microcontroller pause frame detection, TCR*n*[TFC_PAUSE] may remain set and cause the transmission of a single pause frame. In this case, the EIR*n*[GRA] interrupt is not asserted.

26.5.12 Inter-Packet Gap (IPG) Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it is ignored and a collision occurs.

The receiver accepts back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the receiver may discard the following frame.

26.5.13 Collision Managing

If a collision occurs during frame transmission, the Ethernet controller continues the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, a JAM pattern is sent after the end of the preamble sequence.

If a collision occurs within 512 bit times (one slot time), the retry process is initiated. The transmitter waits a random number of slot times. If a collision occurs after 512 bit times, then no retransmission is performed and the end of frame buffer is closed with a Late Collision (LC) error indication.

26.5.14 MII Internal and External Loopback

Internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Internal and external loopback are configured using combinations of the RCR*n*[LOOP, DRT] and TCR*n*[FDEN] bits.

Set FDEN for internal and external loopback.

For internal loopback, set RCR*n*[LOOP] and clear RCR*n*[DRT]. FEC*n*_TXEN and FEC*n*_TXER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the transmit and receive blocks use the internal bus clock instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being DMA'd to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underruns and receive FIFO overflows.

For external loopback, clear RCR*n*[LOOP] and RCR*n*[DRT], and configure the external transceiver for loopback.

26.5.15 RMII Loopback

The FEC also supports RMII loopback. In this mode, transmit data is sent to the RMII receive logic and out the FEC_TXD[1:0] pins. To enable RMII loopback mode, set RCR*n*[RMII_MODE, RMII_LOOP] and TCR*n*[FDEN] and clear RCR*n*[RMII_ECHO, LOOP].

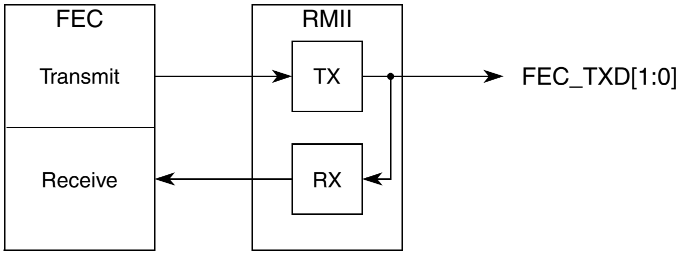


Figure 26-29. RMII Loopback Mode

26.5.16 RMII Echo

The ethernet controller also supports RMII echo mode, which transmits data on FEC_TXD[1:0] as it is received on FEC_RXD[1:0]. To enable RMII echo mode, set RCR*n*[RMII_MODE, RMII_ECHO] bits and clear RCR*n*[RMII_LOOP, LOOP].

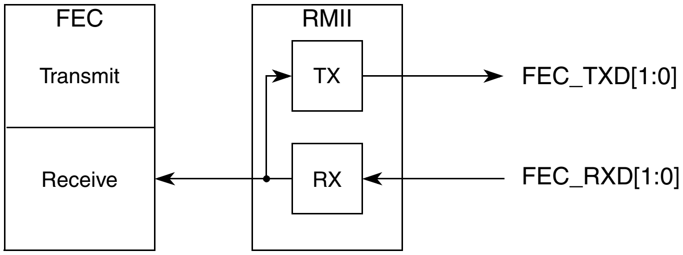


Figure 26-30. RMII Echo Mode

26.5.17 Ethernet Error-Managing Procedure

The Ethernet controller reports frame reception and transmission error conditions using the MIB block counters, the FEC RxBDs, and the EIR*n* register.

26.5.17.1 Transmission Errors

26.5.17.1.1 Transmitter Underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed, and EIR*n*[UN] is set. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame. The UN interrupt is asserted if enabled in the EIMR*n* register.

26.5.17.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and EIR*n*[RL] is set. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame. The RL interrupt is asserted if enabled in the EIMR*n* register.

26.5.17.1.3 Late Collision

When a collision occurs after the slot time (512 bits starting at the Preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and EIR*n*[LC] is set. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame. The LC interrupt is asserted if enabled in the EIMR*n* register.

26.5.17.1.4 Heartbeat

Some transceivers have a self-test called heartbeat or signal quality error. To signify a good self-test, the transceiver indicates a collision to the FEC within four microseconds after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver continues to function properly. This is the heartbeat condition.

If TCR*n*[HBC] is set and the heartbeat condition is not detected by the FEC after a frame transmission, a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets EIR*n*[HB], and generates the HBERR interrupt if it is enabled.

26.5.17.2 Reception Errors

26.5.17.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, FEC sets RxBD*n*[OV]. All subsequent data in the frame is discarded and subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. The driver must discard this frame.

26.5.17.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller manages up to seven dribbling bits when the receive frame terminates past an non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If there is a CRC error, the frame non-octet aligned (NO) error is reported in the RxBD*n*. If there is no CRC error, no error is reported.

26.5.17.2.3 CRC Error

When a CRC error occurs with no dribble bits, FEC closes the buffer and sets RxBD*n*[CR]. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

26.5.17.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes the BABR interrupt is generated, and RxBD*n*[LG] is set. The frame is not truncated unless the frame length exceeds 2047 bytes.

26.5.17.2.5 Truncation

When the receive frame length exceeds 2047 bytes, frame is truncated and RxBD*n*[TR] is set.

Chapter 27
Synchronous Serial Interface (SSI)

27.1 Introduction

This section presents the synchronous serial interface (SSI), and discusses the architecture, the programming model, the operating modes, and initialization of the SSI module.

The SSI module, as shown in Figure 27-1, consists of separate transmit and receive circuits with FIFO registers and separate serial clock and frame sync generation for the transmit and receive sections. The second set of Tx and Rx FIFOs replicates the logic used for the first set of FIFOs.

NOTE

This device contains SSI bits to control the clock rate and the SSI DMA request sources within the chip configuration module (CCM). See Chapter 11, "Chip Configuration Module (CCM)," for detailed information on these bit fields.

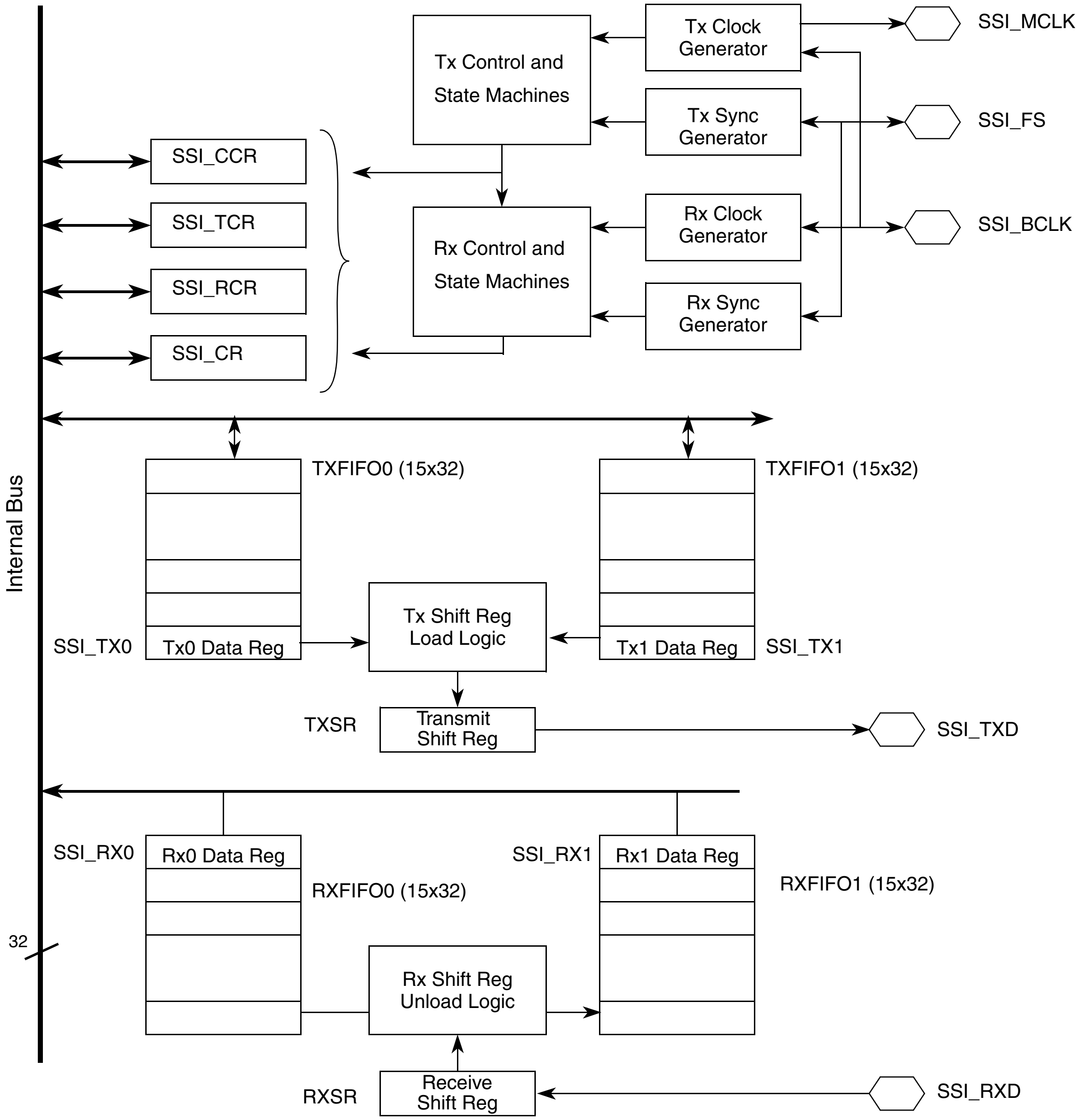


Figure 27-1. SSI Block Diagram

27.1.1 Overview

The SSI is a full-duplex serial port that allows the processor to communicate with a variety of serial devices. Such serial devices are:

- Standard codecs
- Digital signal processors (DSPs)
- Microprocessors
- Peripherals
- Audio codecs that implement the inter-IC sound bus (I^2S) and the Intel® AC97 standards

The SSI module typically transfers samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with shared clock generation and frame synchronization.

NOTE

> The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, “Pin Multiplexing and Control”) prior to configuring the SSI.

27.1.2 Features

The SSI includes the following features:

- Synchronous transmit and receive sections with shared internal/external clocks and frame syncs, operating in master or slave mode.
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with up to 32 time slots
- Gated clock mode operation requiring no frame sync
- Two sets of transmit and receive FIFOs. Each FIFO is 15x32 bits, which can be used in network mode to provide two independent channels for transmission and reception
- Programmable data interface modes such as I^2S, lsb, msb aligned
- Programmable word length (8, 10, 12, 16, 18, 20, 22 or 24 bits)
- Program options for frame sync and clock generation
- Programmable I^2S modes (master or slave). Oversampling clock available as output from SSI_MCLK in I^2S master mode
- AC97 support
- Completely separate clock and frame sync selections. In the AC97 standard, the clock is taken from an external source and frame sync is generated internally.
- Programmable oversampling clock (SSI_MCLK) of the sampling frequency available as output in master mode
- Programmable internal clock divider
- Transmit and receive time slot mask registers for reduced CPU overhead
- SSI power-down feature

27.1.3 Modes of Operation

SSI has the following basic synchronous operating modes.

- Normal mode
- Network mode
- Gated clock mode

These modes can be programmed via the SSI control registers. Table 27-1 lists these operating modes and some of the typical applications in which they can be used:

Table 27-1. SSI Operating Modes

TX, RX Sections	Serial Clock	Mode	Typical Application
Synchronous	Continuous	Normal	Multiple synchronous codecs
Synchronous	Continuous	Network	TDM codec or DSP network
Synchronous	Gated	Normal	SPI-type devices; DSP to MCU

The transmit and receive sections of the SSI are only available in synchronous mode. In this mode, the transmitter and the receiver use a common clock and frame synchronization signal. The SSI_RCR[RXBIT0, RSHFD] bits can continue affecting shifting-in of received data in synchronous mode. Continuous or gated clock mode can be selected. In continuous mode, the clock runs continuously. In gated clock mode, the clock is only functioning during transmission.

Normal or network mode can also be selected. In normal mode, the SSI functions with one data word of I/O per frame. In network mode, any number from two to thirty-two data words of I/O per frame can be used. Network mode is typically used in star or ring time-division-multiplex networks with other processors or codecs, allowing interface to time division multiplexed networks without additional logic. Use of the gated clock is not allowed in network mode. These distinctions result in the basic operating modes that allow the SSI to communicate with a wide variety of devices.

Typically, normal and network modes are used in a periodic manner, where data transfers at regular intervals, such as at the sampling rate of an external codec. Both modes use the concept of a frame. The beginning of the frame is marked with a frame sync when programmed with continuous clock. The SSI_CCR[DC] bits determine length of the frame, depending on whether data is being transmitted or received.

The number of words transferred per frame depends on the mode of the SSI. In normal mode, one data word transfers per frame. In network mode, the frame divides into two to 32 time slots. In each time slot, one data word is optionally transferred.

Apart from the above basic modes of operation, SSI supports the following modes that require some specific programming:

- I^{2}S mode
- AC97 mode
  - AC97 fixed mode
  - AC97 variable mode

In non-I^{2}S slave modes (external frame sync), the SSI's programmed word length setting should be equal to the word length setting of the master. In I^{2}S slave mode, the SSI's programmed word length setting can be lesser than or equal to the word length setting of the I^{2}S master (external codec).

In slave modes, the SSI's programmed frame length setting (DC bits) can be lesser than or equal to the frame length setting of the master (external codec).

See Section 27.4.1, "Detailed Operating Mode Descriptions," for more details on the above modes.

27.2 External Signal Description

The five SSI signals are explained below.

Table 27-2. Signal Properties

Name	Function	Direction	Reset State	Pull up
SSI_CLKIN	SSI Clock Input	I	I	Passive
SSI_BCLK	Serial Bit Clock	I/O	0	Passive
SSI_MCLK	Serial Master Clock	O	0	Passive
SSI_FS	Serial Frame Sync	I/O	0	Passive
SSI_RXD	Serial Receive Data	I	—	—
SSI_TXD	Serial Transmit Data	O	0	Passive

27.2.1 SSI_CLKIN — SSI Clock Input

The SSI module can be clocked by the internal core frequency derived from the PLL or this input clock. The source is selected by the MISCCR[SSISRC] bit in the CCM. See Chapter 11, “Chip Configuration Module (CCM),” and Figure 27-37.

27.2.2 SSI_BCLK — Serial Bit Clock

This input or output signal is used by the transmitter and receiver and can be continuous or gated. During gated clock mode, data on the SSI_BCLK port is valid only during the transmission of data; otherwise, it is pulled to the programmed inactive state.

27.2.3 SSI_MCLK — Serial Master Clock

This clock signal is output from the device when it is the master. When in I^2S master mode, this signal is referred to as the oversampling clock. The frequency of SSI_MCLK is a multiple of the frame clock.

27.2.4 SSI_FS — Serial Frame Sync

The input or output frame sync is used by the transmitter and receiver to synchronize the transfer of data. The frame sync signal can be one bit or one word in length and can occur one bit before the transfer of data or right at the transfer of data. In gated clock mode, the frame sync signal is not used. If SSI_FS is configured as an input, the external device should drive SSI_FS during rising edge of SSI_BCLK.

27.2.5 SSI_RXD — Serial Receive Data

The SSI_RXD port is an input and brings serial data into the receive data shift register.

27.2.6 SSI_TXD — Serial Transmit Data

The SSI_TXD port is an output and transmits data from the serial transmit shift register. The SSI_TXD port is an output port when data is transmitted and disabled between data word transmissions and on the trailing edge of the bit clock after the last bit of a word is transmitted.

Figure 27-2 shows the main SSI configurations. These ports support all transmit and receive functions with continuous or gated clock as shown. Gated clock implementations do not require the use of the frame sync port (SSI_FS).

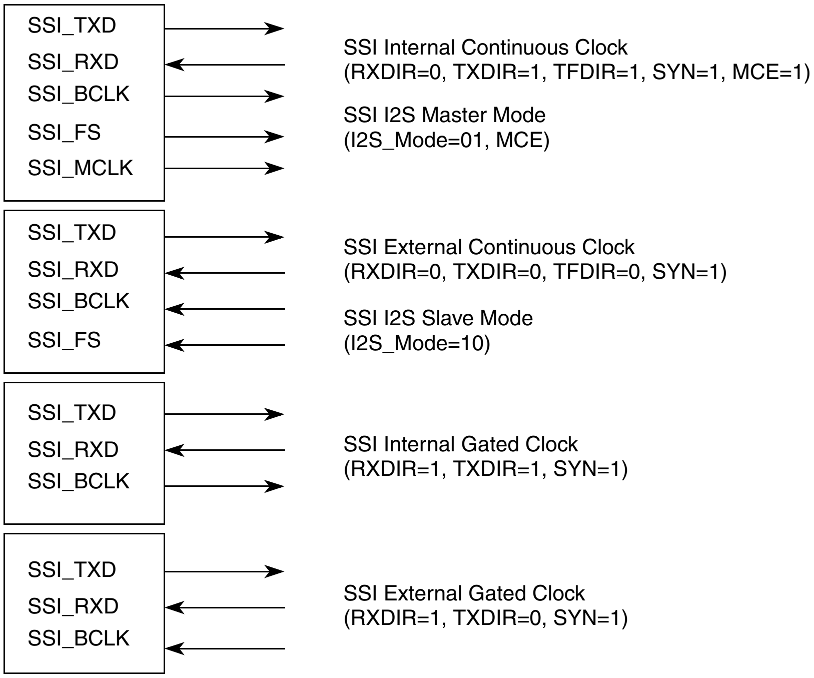


Figure 27-2. Synchronous SSI Configurations—Continuous and Gated Clock

Figure 27-3 shows an example of the port signals for an 8-bit data transfer. Continuous and gated clock signals are shown, as well as the bit-length frame sync signal and the word-length frame sync signal. The shift direction can be defined as msb first or lsb first, and there are other options on the clock and frame sync.

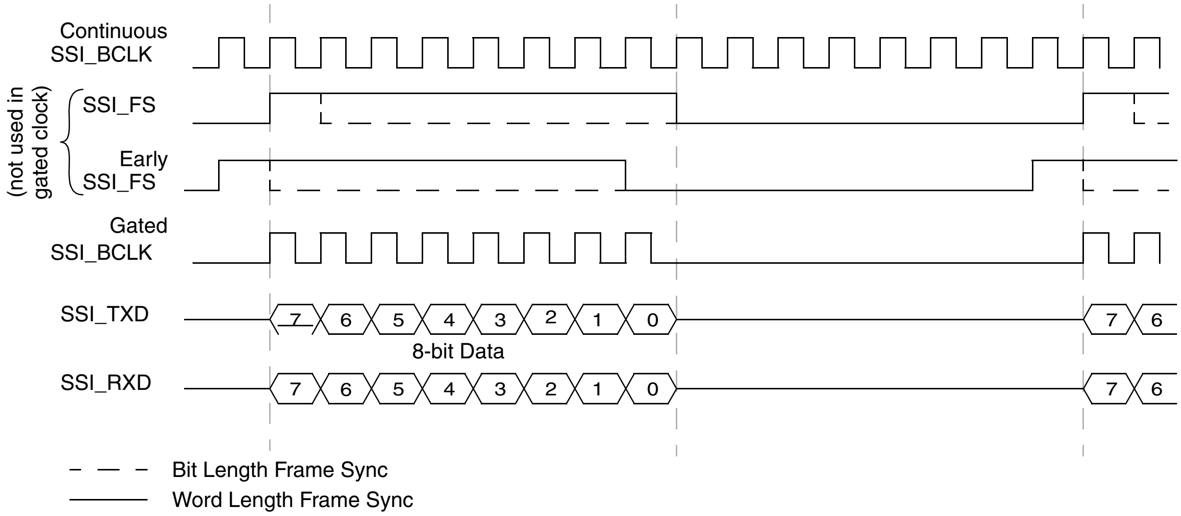


Figure 27-3. Serial Clock and Frame Sync Timing

Table 27-3. Clock and Frame Sync Pin Configuration

SSI_CR [SYN]	SSI_RCR [RXDIR]	SSI_TCR		SSI_BCLK	SSI_FS
		TXDIR	TFDIR		
Synchronous Mode					
1	0	0	0	Bit clock in	FS in
1	0	0	1	Bit clock in	FS out
1	0	1	0	Bit clock out	FS in
1	0	1	1	Bit clock out	FS out
1	1	0	x	Gated clock in	—
1	1	1	x	Gated clock out	—

27.3 Memory Map/Register Definition

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

Table 27-4. SSI Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_C000	SSI Transmit Data Register 0 (SSI_TX0)	32	R/W	0x0000_0000	27.3.1/27-8
0xFC0B_C004	SSI Transmit Data Register 1 (SSI_TX1)	32	R/W	0x0000_0000	27.3.1/27-8
0xFC0B_C008	SSI Receive Data Register 0 (SSI_RX0)	32	R	0x0000_0000	27.3.4/27-10
0xFC0B_C00C	SSI Receive Data Register 1 (SSI_RX1)	32	R	0x0000_0000	27.3.4/27-10
0xFC0B_C010	SSI Control Register (SSI_CR)	32	R/W	0x0000_0000	27.3.7/27-13
0xFC0B_C014	SSI Interrupt Status Register (SSI_ISR)	32	R	0x0000_3003	27.3.8/27-15

Table 27-4. SSI Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_C018	SSI Interrupt Enable Register (SSI_IER)	32	R/W	0x0000_3003	27.3.9/27-20
0xFC0B_C01C	SSI Transmit Configuration Register (SSI_TCR)	32	R/W	0x0000_0200	27.3.10/27-21
0xFC0B_C020	SSI Receive Configuration Register (SSI_RCR)	32	R/W	0x0000_0200	27.3.11/27-23
0xFC0B_C024	SSI Clock Control Register (SSI_CCR)	32	R/W	0x0004_0000	27.3.12/27-24
0xFC0B_C02C	SSI FIFO Control/Status Register (SSI_FCSR)	32	R/W	0x0081_0081	27.3.13/27-25
0xFC0B_C038	SSI AC97 Control Register (SSI_ACR)	32	R/W	0x0000_0000	27.3.14/27-32
0xFC0B_C03C	SSI AC97 Command Address Register (SSI_ACADD)	32	R/W	0x0000_0000	27.3.15/27-33
0xFC0B_C040	SSI AC97 Command Data Register (SSI_ACDAT)	32	R/W	0x0000_0000	27.3.16/27-33
0xFC0B_C044	SSI AC97 Tag Register (SSI_ATAG)	32	R/W	0x0000_0000	27.3.17/27-34
0xFC0B_C048	SSI Transmit Time Slot Mask Register (SSI_TMASK)	32	R/W	0x0000_0000	27.3.18/27-34
0xFC0B_C04C	SSI Receive Time Slot Mask Register (SSI_RMASK)	32	R/W	0x0000_0000	27.3.19/27-35

27.3.1 SSI Transmit Data Registers 0 and 1 (SSI_TX0/1)

The SSI_TX0/1 registers store the data to be transmitted by the SSI. For details on data alignment see Section 27.4.4, “Supported Data Alignment Formats.”

Address: 0xFC0B_C000 (SSI_TX0)
0xFC0B_C004 (SSI_TX1)

Access: User 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
R	SSI_TX																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-4. SSI Transmit Data Registers (SSI_TX0, SSI_TX1)

Table 27-5. SSI_TX0/1 Field Descriptions

Field	Description
31–0 SSI_TX	SSI transmit data. The SSI_TX0/1 registers are implemented as the first word of their respective Tx FIFOs. Data written to these registers transfers to the transmit shift register (TXSR), when shifting of the previous data is complete. If both FIFOs are in use, data alternately transfers from SSI_TX0 and SSI_TX1 to TXSR. SSI_TX1 can only be used in two-channel mode. Multiple writes to the SSI_TX registers do not result in the previous data being over-written by the subsequent data. Instead, they are ignored. Protection from over-writing is present irrespective of whether the transmitter is enabled or not. Example: If Tx FIFO0 is in use and you write Data1 – 16 to SSI_TX0, Data16 does not overwrite Data1. Data1 – 15 are stored in the FIFO while Data16 is discarded. Example: If Tx FIFO0 is not in use and you write Data1, Data2 to SSI_TX0, Data2 does not overwrite Data1 and is discarded. **Note:** Enable SSI (SSI_CR[SSI_EN] = 1) before writing to the SSI transmit data registers

27.3.2 SSI Transmit FIFO 0 and 1 Registers

The SSI transmit FIFO registers are 15x32-bit registers. These registers are not directly accessible. The transmit shift register (TXSR) receives its values from these FIFO registers. When the transmit interrupt enable (SSI_IER[TIE]) bit and either of the transmit FIFO empty (SSI_ISR[TFE0, TFE1]) bits are set, an interrupt is generated when the data level in of the SSI transmit FIFOs falls below the selected threshold.

27.3.3 SSI Transmit Shift Register (TXSR)

TXSR is a 24-bit shift register that contains the data transmitted and is not directly accessible. When a continuous clock is used, the selected bit clock shifts data out to the SSI_TXD pin when the associated frame sync is asserted. When a gated clock is used, the selected gated clock shifts data out to the SSI_TXD port.

The word length control bits (SSI_CCR[WL]) determine the number of bits to shift out of the TXSR before it is considered empty and can be written to again. The data to be transmitted occupies the most significant portion of the shift register if SSI_TCR[TXBIT0] is cleared. Otherwise, it occupies the least significant portion. The unused portion of the register is ignored.

NOTE

> If TXBIT0 is cleared and the word length is less than 16 bits, data occupies the most significant portion of the lower 16 bits of the transmit register.

When SSI_TCR[SHFD] is cleared, data is shifted out of this register with the most significant bit (msb) first. If this bit is set, the least significant bit (lsb) is shifted out first. The following figures show the transmitter loading and shifting operation. They illustrate some possible values for WL, which can be extended for the other values.

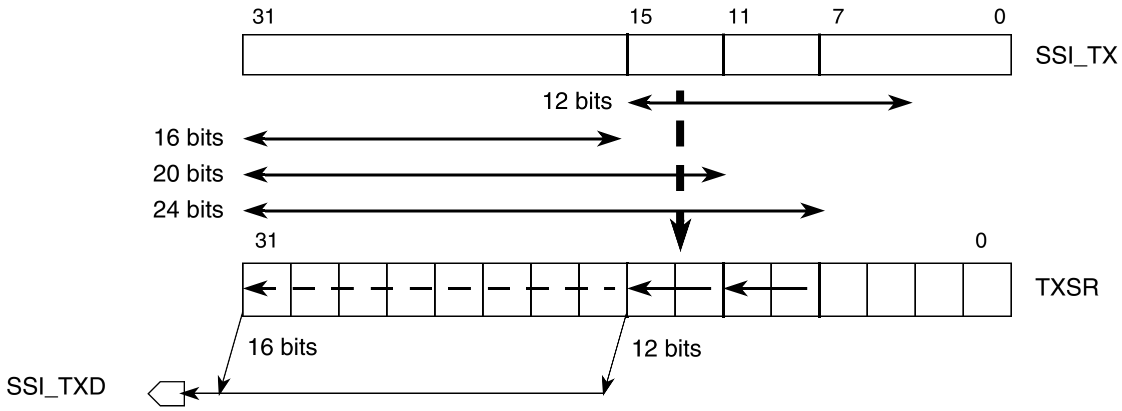


Figure 27-5. Transmit Data Path (TXBIT0=0, TSHFD=0) (msb Alignment)

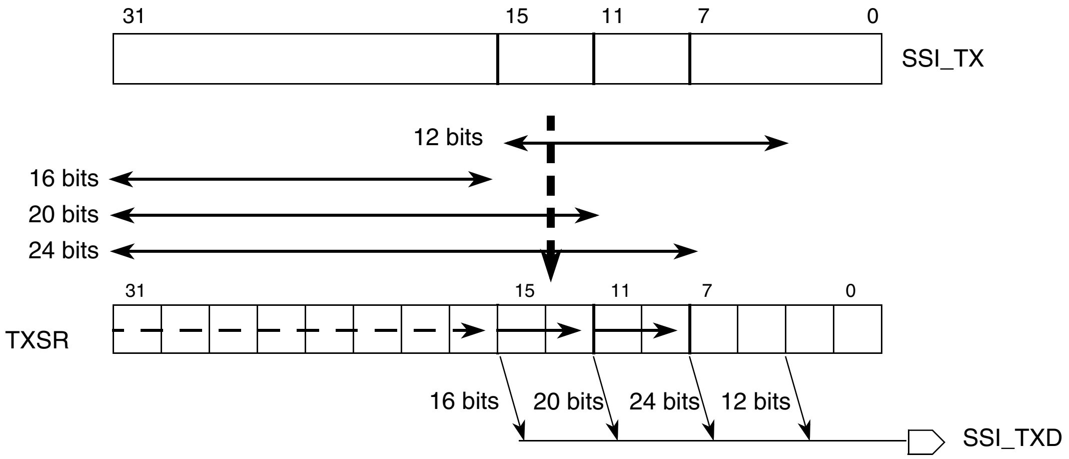


Figure 27-6. Transmit Data Path (TXBIT0=0, TSHFD=1) (msb Alignment)

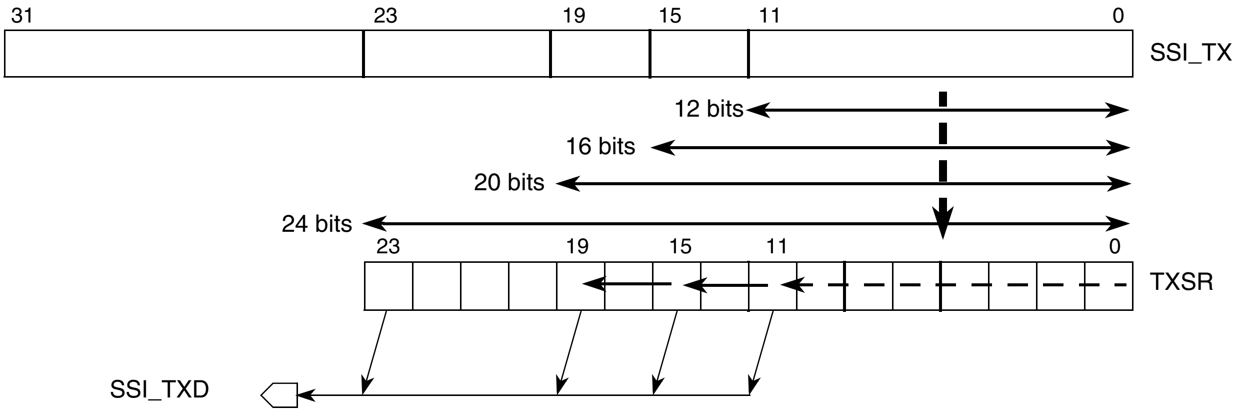


Figure 27-7. Transmit Data Path (TXBIT0=1, TSHFD=0) (lsb Alignment)

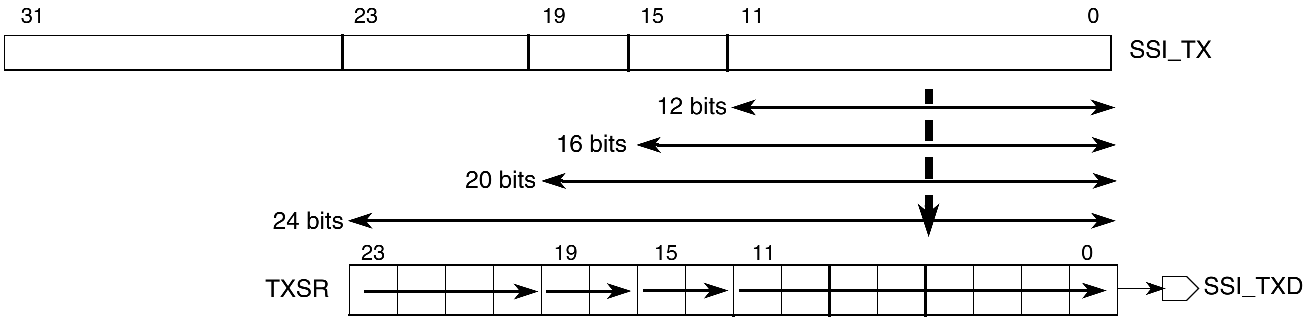


Figure 27-8. Transmit Data Path (TXBIT0=1, TSHFD=1) (lsb Alignment)

27.3.4 SSI Receive Data Registers 0 and 1 (SSI_RX0/1)

The SSI_RX0/1 registers store the data received by the SSI. For details on data alignment see Section 27.3.6, “SSI Receive Shift Register (RXSR).”

Address: 0xFC0B_C008 (SSI_RX0)
0xFC0B_C00C (SSI_RX1)

Access: User 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
R	SSI_RX																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-9. SSI Receive Data Registers (SSI_RX0, SSI_RX1)

Table 27-6. SSI_RX0/1 Field Descriptions

Field	Description
31–0 SSI_RX	SSI receive data. SSI_RX0/1 are implemented as the first word of their respective Rx FIFOs. These bits receive data from RXSR depending on the mode of operation. If both FIFOs are in use, data is transferred to each data register alternately. SSI_RX1 is only used in two-channel mode.

27.3.5 SSI Receive FIFO 0 and 1 Registers

The SSI receive FIFO registers are 15x32-bit registers and are not directly accessible. They always accept data from the receive shift register (RXSR). If the associated interrupt is enabled, an interrupt is generated when the data level in either of the SSI receive FIFOs reaches the selected threshold.

27.3.6 SSI Receive Shift Register (RXSR)

RXSR is a 24-bit shift register receiving incoming data from the SSI_RXD pin. This register is not directly accessible. When a continuous clock is used, data is shifted in by the bit clock when the associated frame sync is asserted. When a gated clock is used, data is shifted in by the gated clock. Data is assumed to be received msb first if SSI_RCR[SHFD] is cleared. If this bit is set, the data is received lsb first. Data is transferred to the appropriate SSI receive data register or receive FIFOs (if the receive FIFO is enabled and the corresponding SSI_RX is full) after a word has been shifted in. For receiving less than 24 bits of data, the lsb bits are appended with 0.

The following figures show the receiver loading and shifting operation. They illustrate some possible values for WL, which can be extended for the other values.

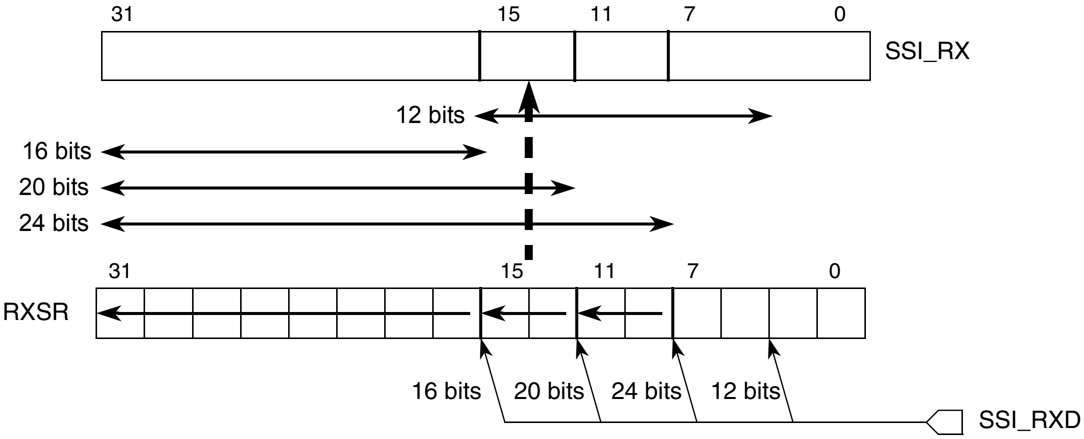


Figure 27-10. Receive Data Path (RXBIT0=0, RSHFD=0) (msb Alignment)

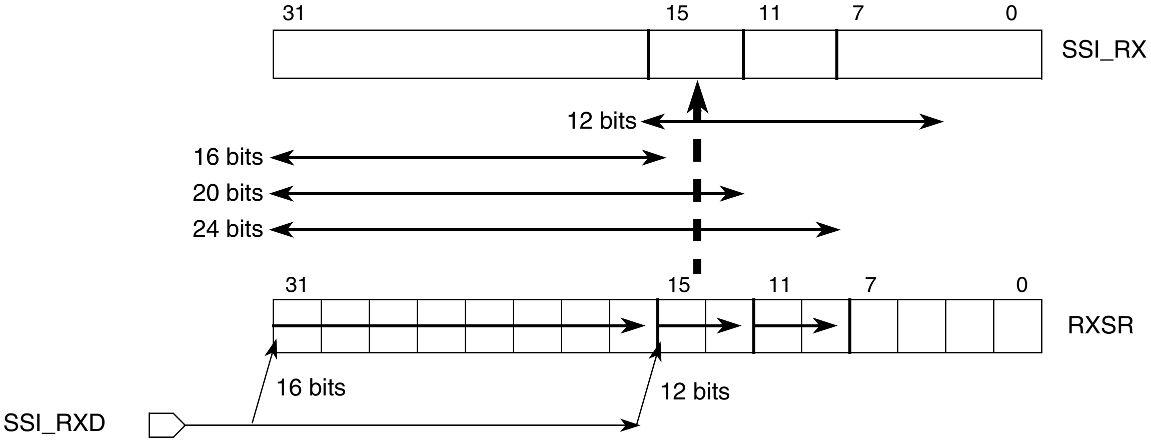


Figure 27-11. Receive Data Path (RXBIT0=0, RSHFD=1) (msb Alignment)

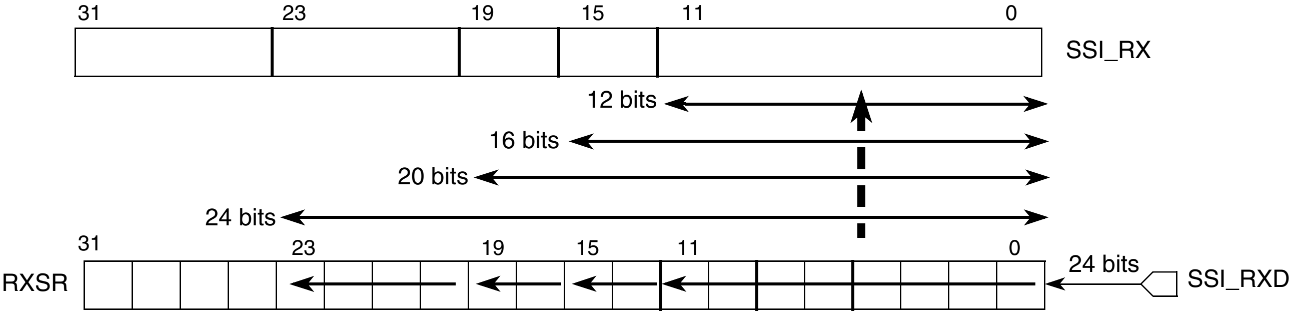


Figure 27-12. Receive Data Path (RXBIT0=1, RSHFD=0) (lsb Alignment)

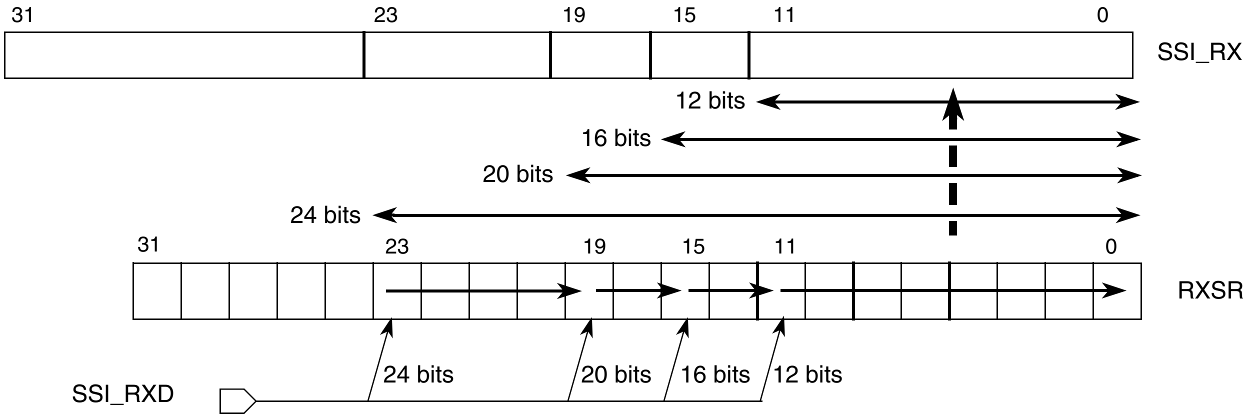


Figure 27-13. Receive Data Path (RXBIT0=1, RSHFD=1) (lsb Alignment)

27.3.7 SSI Control Register (SSI_CR)

The SSI control register sets up the SSI modules. SSI operating modes are selected in this register (except AC97 mode, which is selected in SSI_ACR register).

Address: 0xFC0B_C010 (SSI_CR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	CIS	TCH	MCE	I2S		SYN	NET	RE	TE	SSI _EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-14. SSI Control Register (SSI_CR)

Table 27-7. SSI_CR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9 CIS	Clock idle state. Controls the idle state of the transmit clock port (SSI_BCLK and SSI_MCLK) during internal gated clock mode. 0 Clock idle state is 1 1 Clock idle state is 0
8 TCH	Two channel operation enable. In this mode, two time slots are used out of the possible 32. Any two time slots (0 – 31) can be selected by the mask registers. The data in the two time slots is alternately handled by the two data registers (0 and 1). While receiving, RXSR transfers data to SSI_RX0 and SSI_RX1 alternately, and while transmitting, data is alternately transferred from SSI_TX0 and SSI_TX1 to TXSR. Two channel operation can be enabled for an even number of slots larger than two to optimize usage of both FIFOs. However, TCH should be cleared for an odd number of time slots. 0 Two channel mode disabled 1 Two channel mode enabled

Table 27-7. SSI_CR Field Descriptions (continued)

Field	Description
7 MCE	Master clock enable. Allows the SSI to output the master clock at the SSI_MCLK port, if network mode and transmit internal clock mode are set. The DIV2, PSR, and PM bits determine the relationship between the bit clock (SSI_BCLK) and SSI_MCLK. In I^{2}S master mode, this bit is used to output the oversampling clock on SSI_MCLK. 0 Master clock not output on the SSI_MCLK pin 1 Master clock output on the SSI_MCLK pin
6–5 I2S	I^{2}S mode select. Selects normal, I^{2}S master, or I^{2}S slave mode. Refer to Section 27.4.1.4, "I2S Mode," for a detailed description of I^{2}S mode. 00 Normal mode 01 I^{2}S master mode 10 I^{2}S slave mode 11 Normal mode
4 SYN	Synchronous mode enable. In synchronous mode, transmit and receive sections of SSI share a common clock port (SSI_BCLK) and frame sync port (SSI_FS). 0 Reserved. 1 Synchronous mode selected.
3 NET	Network mode enable. 0 Network mode not selected 1 Network mode selected
2 RE	Receiver enable. When this bit is set, data reception starts with the arrival of the next frame sync. If data is received when this bit is cleared, data reception continues with the end of the current frame and then stops. If this bit is set again before the second to last bit of the last time slot in the current frame, reception continues without interruption. 0 Receiver disabled 1 Receiver enabled
1 TE	Transmitter. Enables the transfer of the contents of the SSI_TX registers to the TXSR, and also enables the internal transmit clock. The transmit section is enabled when this bit is set and a frame boundary is detected. When this bit is cleared, the transmitter continues to send data until the end of the current frame and then stops. Data can be written to the SSI_TX registers with the TE bit cleared (the corresponding TDE bit is cleared). If the TE bit is cleared and set again before the second to last bit of the last time slot in the current frame, data transmission continues without interruption. The normal transmit enable sequence is to: 1. Write data to the SSI_TX register(s) 2. Set the TE bit The normal transmit disable sequence is to: 1. Wait for TDE to set 2. Clear the TE and TIE bits In gated clock mode, clearing the TE bit results in the clock stopping after the data currently in TXSR has shifted out. When the TE bit is set, the clock starts immediately in internal gated clock mode. 0 Transmitter disabled 1 Transmitter enabled
0 SSI_EN	SSI enable. When disabled, all SSI status bits are reset to the same state produced by the power-on reset, all control bits are unaffected, and the contents of Tx and Rx FIFOs are cleared. When SSI is disabled, all internal clocks are disabled (except the register access clock). 0 SSI module is disabled 1 SSI module is enabled

27.3.8 SSI Interrupt Status Register (SSI_ISR)

The SSI interrupt status register monitors the SSI. This register is read-only and is used by the processor to interrogate the status of the SSI module. All receiver-related interrupts are generated only if the receiver is enabled (SSI_CR[RE] = 1). Likewise, all transmitter-related interrupts are generated only if the transmitter is enabled (SSI_CR[TE] = 1).

NOTE

Refer to Section 27.4.5, "Receive Interrupt Enable Bit Description," and Section 27.4.6, "Transmit Interrupt Enable Bit Description," for more details on SSI interrupt generation.

All flags in the SSI_ISR are updated after the first bit of the next SSI word has completed transmission or reception. Some status bits (ROE0/1 and TUE0/1) are cleared by reading the SSI_ISR followed by a read or write to the SSI_RX0/1 or SSI_TX0/1 registers.

Address: 0xFC0B_C014 (SSI_ISR) Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	CMDAU	CMDDU	RXT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RDR1	RDR0	TDE1	TDE0	ROE1	ROE0	TUE1	TUE0	TFS	RFS	TLS	RLS	RFF1	RFF0	TFE1	TFE0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 27-15. SSI Interrupt Status Register (SSI_ISR)

Table 27-8. SSI_ISR Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18 CMDAU	AC97 command address register updated. This bit causes the command address updated interrupt when the SSI_IER[CMDAU] bit is set. This status bit is set each time there is a difference in the previous and current value of the received command address. This bit is cleared upon reading the SSI_ACADD register. 0 No change in SSI_ACADD register 1 SSI_ACADD register updated with different value
17 CMDU	AC97 command data register updated. This bit causes the command data updated interrupt when the SSI_IER[CMDDU] bit is set. This status bit is set each time there is a difference in the previous and current value of the received command data. This bit is cleared upon reading the SSI_ACDAT register. 0 No change in SSI_ACDAT register 1 SSI_ACDAT register updated with different value
16 RXT	AC97 receive tag updated. This status bit is set each time there is a difference in the previous and current value of the received tag. It causes the receive tag interrupt if the SSI_IER[RXT] bit is set. This bit is cleared upon reading the SSI_ATAG register. 0 No change in SSI_ATAG register 1 SSI_ATAG register updated with different value

Table 27-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15
RDR1</td><td>Receive data ready 1. Only valid in two-channel mode. Indicates new data is available for the processor to read.
<table>
<tr><th rowspan="2">Rx FIFO1</th><th colspan="2">Receive data 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[RIE] set
• SSI_IER[RFF1] set</td><td>• SSI_ISR[RFF1] sets</td></tr>
<tr><td>Disabled</td><td>• SSI_IER[RIE] set
• SSI_IER[RDR1] set</td><td>• SSI_RX1 loaded with new value</td></tr>
</table>
<table>
<tr><th>Rx FIFO1</th><th>RDR1 is set when</th><th>RDR1 is cleared during any of the following</th></tr>
<tr><td>Enabled</td><td>• Rx FIFO1 loaded with new value</td><td>• Rx FIFO1 is empty
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td><td>• SSI_RX1 loaded with new value</td><td>• SSI_RX1 is read
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
<tr><td>14
RDR0</td><td>Receive data ready 0. Similar description as RDR1 but pertains to Rx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 No new data for core to read
1 New data for core to read</td></tr>
<tr><td>13
TDE1</td><td>Transmit data register empty 1. Only valid in two-channel mode. Indicates that data needs to be written to the SSI.
<table>
<tr><th rowspan="2">Tx FIFO1</th><th colspan="2">Transmit data 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[TIE] set
• SSI_IER[TDE1] set</td><td>• SSI_ISR[TDE1] sets</td></tr>
<tr><td>Disabled</td><td>• SSI_IER[TIE] set
• SSI_IER[TDE1] set</td><td>• SSI_TX1 data transferred to TXSR</td></tr>
</table>
<table>
<tr><th>Tx FIFO1</th><th>TDE1 is set when</th><th>TDE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• At least one empty slot in Tx FIFO1</td><td>• Tx FIFO1 is full
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td><td>• SSI_TX1 data transferred to TXSR</td><td>• SSI_TX1 is written
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
</table>

Table 27-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>12
TDE0</td><td>Transmit data register empty 0. Similar description as TE1 but pertains to Tx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 Data available for transmission
1 Data needs to be written by the core for transmission</td></tr>
<tr><td>11
ROE1</td><td>Receiver overrun error 1. Only valid in two-channel mode. Indicates an overrun error has occurred.
<table>
<tr><th rowspan="2">Rx FIFO1</th><th colspan="2">Receiver overrun error 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td rowspan="2">• SSI_IER[RIE] set
• SSI_IER[ROE1] set</td><td rowspan="2">• SSI_ISR[ROE1] sets</td></tr>
<tr><td>Disabled</td></tr>
</table>
<table>
<tr><th>Rx FIFO1</th><th>ROE1 is set when all of the following occur</th><th>ROE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• RXSR is full
• Rx FIFO1 is full</td><td rowspan="2">• Reading SSI_ISR when ROE1 is set
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td><td>• RXSR is full
• SSI_RX1 is full</td></tr>
</table>
<b>Note:</b> If Rx FIFO 1 is enabled, the RFF1 flag indicates the FIFO is full.
If Rx FIFO 1 is disabled, the RDR1 flag indicates the SSI_RX1 register is full.</td></tr>
<tr><td>10
ROE0</td><td>Receiver overrun error 0. Similar description as ROE1 but pertains to Rx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 No overrun detected
1 Receiver 0 overrun error occurred</td></tr>
<tr><td>9
TUE1</td><td>Transmitter underrun error 1. Only valid in two-channel mode. When a transmit underrun error occurs, the previous data is retransmitted. In network mode, each time slot requires data transmission (unless masked through the SSI_TMASK register), when the transmitter is enabled.
<table>
<tr><th rowspan="2">Tx FIFO1</th><th colspan="2">Transmit underrun error 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td rowspan="2">• SSI_IER[TIE] set
• SSI_IER[TUE1] set</td><td rowspan="2">• SSI_ISR[TUE1] sets</td></tr>
<tr><td>Disabled</td></tr>
</table>
<table>
<tr><th>Tx FIFO1</th><th>TUE1 is set when all of the following occur</th><th>TUE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td rowspan="2">• TXSR is empty
• SSI_ISR[TDE1] set
• Transmit time slot occurs</td><td rowspan="2">• Reading SSI_ISR when TUE1 is set
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td></tr>
</table>
</td></tr>
</table>

Table 27-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8
TUE0</td><td>Transmitter underrun error 0. Similar description as TUE1 but pertains to TDE0 and it is not necessary to be in two-channel mode for this bit to be set.
0 No underrun detected
1 Transmitter 0 underrun error occurred</td></tr>
<tr><td>7
TFS</td><td>Transmit frame sync. Indicates occurrence of a transmit frame sync during transmission of the last word written to the SSI_TX registers
<table>
<tr><th rowspan="2">SSI Mode</th><th colspan="2">Transmit frame sync interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Normal</td><td rowspan="2">• SSI_IER[TIE] set
• SSI_IER[TFS] set</td><td rowspan="2">• SSI_ISR[TFS] sets</td></tr>
<tr><td>Network</td></tr>
</table>
<table>
<tr><th>SSI Mode</th><th>TFS is set when</th><th>TFS is cleared when any of the following occur</th></tr>
<tr><td>Normal</td><td>• TFS is always set</td><td>• SSI reset
• POR reset</td></tr>
<tr><td>Network</td><td>• First time slot transmission</td><td>• Starts transmitting next time slot
• SSI reset
• POR reset</td></tr>
</table>
<b>Note:</b> Data written to the SSI_TX registers during the time slot when the TFS flag is set is sent during the second time slot (in network mode) or in the next first time slot (in normal mode).</td></tr>
<tr><td>6
RFS</td><td>Receive frame sync. Indicates occurrence of a receive frame sync during reception of the next word in SSI_RX registers.
<table>
<tr><th rowspan="2">SSI Mode</th><th colspan="2">Receive frame sync interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Normal</td><td rowspan="2">• SSI_IER[RIE] set
• SSI_IER[RFS] set</td><td rowspan="2">• SSI_ISR[RFS] sets</td></tr>
<tr><td>Network</td></tr>
</table>
<table>
<tr><th>SSI Mode</th><th>RFS is set when</th><th>RFS is cleared when any of the following occur</th></tr>
<tr><td>Normal</td><td>• RFS is always set</td><td>• SSI reset
• POR reset</td></tr>
<tr><td>Network</td><td>• First time slot received</td><td>• Starts receiving next time slot
• SSI reset
• POR reset</td></tr>
</table></td></tr>
</table>

Table 27-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
TLS
4
RLS</td><td>Transmit/receive last time slot. Indicates the current time slot is the last time slot of the frame.
<table>
<tr><th></th><th colspan="2">Last time slot interrupts</th></tr>
<tr><th></th><th>Required conditions</th><th>Trigger</th></tr>
<tr><td><b>TLS</b></td><td>• SSI_IER[TIE] set
• SSI_IER[TLS] set</td><td>• SSI_ISR[TLS] sets</td></tr>
<tr><td><b>RLS</b></td><td>• SSI_IER[RIE] set
• SSI_IER[RLS] set</td><td>• SSI_ISR[RLS] sets</td></tr>
</table>
<table>
<tr><th></th><th>Is set when</th><th>Is cleared when
any of the following occur</th></tr>
<tr><td><b>TLS</b></td><td>• Start of last transmit time slot</td><td>• SSI_ISR is read with TLS set
• SSI reset
• POR reset</td></tr>
<tr><td><b>RLS</b></td><td>• End of last receive time slot</td><td>• SSI_ISR is read with RLS set
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
<tr><td>3
RFF1</td><td>Receive FIFO full 1. Only valid in two-channel mode and if Rx FIFO 1 is enabled. When Rx FIFO1 is full, all further data received (for storage in this FIFO) is ignored until the FIFO contents are read.
<table>
<tr><th>Rx FIFO1</th><th colspan="2">Receive FIFO full 1 interrupt</th></tr>
<tr><th></th><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[RIE] set
• SSI_IER[RFF1] set</td><td>• SSI_ISR[RFF1] sets</td></tr>
</table>
<table>
<tr><th>Rx FIFO1</th><th>RFF1 is set when</th><th>RFF1 is cleared when
any of the following occur</th></tr>
<tr><td>Enabled</td><td>• Rx FIFO 1 level reaches Rx FIFO watermark 1 (RFWM1) threshold</td><td>• Rx FIFO 1 level falls below RFWM1 level
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
<tr><td>2
RFF0</td><td>Receive FIFO full 0. Similar to description of RFF1, but pertains to Rx FIFO 0 and is not necessary to be in two-channel mode for this bit to be set.
0 Space available in receive FIFO 0
1 Receive FIFO 0 is full</td></tr>
</table>

Table 27-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>1
TFE1</td><td>Transmit FIFO empty 1. Only valid when in two-channel mode and Tx FIFO 1 is enabled.
<table>
<tr><th rowspan="2">Tx FIFO1</th><th colspan="2">Transmit FIFO full 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[RIE] set
• SSI_IER[TFE1] set</td><td>• SSI_ISR[TFE1] sets</td></tr>
</table>
<table>
<tr><th>Tx FIFO1</th><th>TFE1 is set when any of the following occur</th><th>TFE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• Tx FIFO 1 level falls below Tx FIFO watermark 1 (TFWM1) threshold
• SSI reset
• POR reset</td><td>• Tx FIFO 1 level is more than TFWM1 level</td></tr>
</table>
</td></tr>
<tr><td>0
TFE0</td><td>Transmit FIFO empty 0. Similar to description of TFE1 but pertains to TX FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 Transmit FIFO 0 has data for transmission
1 Transmit FIFO 0 is empty</td></tr>
</table>

27.3.9 SSI Interrupt Enable Register (SSI_IER)

The SSI_IER register sets up the SSI interrupts and DMA requests.

Address: 0xFC0B_C018 (SSI_IER) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	RDMAE	RIE	TDMAE	TIE	CMD AU	CMDU	RXT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RDR1	RDR0	TDE1	TDE0	ROE1	ROE0	TUE1	TUE0	TFS	RFS	TLS	RLS	RFF1	RFF0	TFE1	TFE0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 27-16. SSI Interrupt Enable Register (SSI_IER)

Table 27-9. SSI_IER Field Descriptions

Field	Description
31–23	Reserved, must be cleared.
22 RDMAE	Receive DMA enable. • If the Rx FIFO is enabled, a DMA request generates when either of the SSI_ISR[RFF0/1] bits is set. • If the Rx FIFO is disabled, a DMA request generates when either of the SSI_ISR[RDR0/1] bits is set. 0 SSI receiver DMA requests disabled. 1 SSI receiver DMA requests enabled.
21 RIE	Receive interrupt enable. Allows the SSI to issue receiver related interrupts to the processor. Refer to Section 27.4.5, “Receive Interrupt Enable Bit Description,” for a detailed description of this bit. 0 SSI receiver interrupt requests disabled. 1 SSI receiver interrupt requests enabled.
20 TDMAE	Transmit DMA enable. • If the Tx FIFO is enabled, a DMA request generates when either of the SSI_ISR[TFE0/1] bits is set. • If the Tx FIFO is disabled, a DMA request generates when either of the SSI_ISR[TDE0/1] bits is set. 0 SSI transmitter DMA requests disabled. 1 SSI transmitter DMA requests enabled.
19 TIE	Transmit interrupt enable. Allows the SSI to issue transmitter data related interrupts to the core. Refer to Section 27.4.6, “Transmit Interrupt Enable Bit Description,” for a detailed description of this bit. 0 SSI transmitter interrupt requests disabled. 1 SSI transmitter interrupt requests enabled.
18–0	Controls if the corresponding status bit in SSI_ISR can issue an interrupt to the processor. See Section 27.3.8, “SSI Interrupt Status Register (SSI_ISR),” for details on the individual bits. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.

27.3.10 SSI Transmit Configuration Register (SSI_TCR)

The SSI transmit configuration register directs the transmit operation of the SSI. A power-on reset clears all SSI_TCR bits. However, an SSI reset does not affect the SSI_TCR bits.

Address: 0xFC0B_C01C (SSI_TCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	TX BIT0	TFEN1	TFEN0	TFDIR	TXDIR	TSHFD	TSCKP	TFSI	TFSL	TEFS
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 27-17. SSI Transmit Configuration Register (SSI_TCR)

Table 27-10. SSI_TCR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9 TXBIT0	Transmit bit 0 (Alignment). Allows SSI to transmit data word from bit position 0 or 15/31 in the transmit shift register. The shifting data direction can be msb or lsb first, controlled by the TSHFD bit. 0 msb-aligned. Shift with respect to bit 31 (if the word length is 16, 18, 20, 22 or 24) or bit 15 (if the word length is 8, 10 or 12) of the transmit shift register 1 lsb-aligned. Shift with respect to bit 0 of the transmit shift register
8 TFEN1	Transmit FIFO enable 1. • When enabled, the FIFO allows eight samples to be transmitted by the SSI (per channel) (a ninth sample can be shifting out) before SSI_ISR[TDE1] is set. • When the FIFO is disabled, SSI_ISR[TDE1] is set when a single sample is transferred to the transmit shift register. This issues an interrupt if the interrupt is enabled. 0 Transmit FIFO 1 disabled 1 Transmit FIFO 1 enabled
7 TFEN0	Transmit FIFO enable 0. Similar description as TFEN1, but pertains to Tx FIFO 0. 0 Transmit FIFO 0 disabled 1 Transmit FIFO 0 enabled
6 TFDIR	Frame sync direction. Controls the direction and source of the frame sync signal on the SSI_FS pin. 0 Frame sync is external 1 Frame sync generated internally
5 TXDIR	Clock direction. Controls the direction and source of the clock signal on the SSI_BCLK pin. Refer to Table 27-3 for details of clock port configuration. 0 Clock is external 1 Clock generated internally
4 TSHFD	Transmit shift direction. Controls whether the msb or lsb is transmitted first in a sample. 0 Data transmitted msb first 1 Data transmitted lsb first
3 TSCKP	Transmit clock polarity. Controls which bit clock edge is used to clock out data for the transmit section. 0 Data clocked out on rising edge of bit clock 1 Data clocked out on falling edge of bit clock
2 TFSI	Transmit frame sync invert. Controls the active state of the frame sync I/O signal for the transmit section of SSI. 0 Transmit frame sync is active high 1 Transmit frame sync is active low
1 TFSL	Transmit frame sync length. Controls the length of the frame sync signal generated or recognized for the transmit section. The length of a word-long frame sync is the same as the length of the data word selected by SSI_CCR[WL]. 0 Transmit frame sync is one-word long 1 Transmit frame sync is one-bit-clock-period long
0 TEFS	Transmit early frame sync. Controls when the frame sync is initiated for the transmit section. The frame sync signal is deasserted after one bit for a bit length frame sync (TFSL = 1) and after one word for word length frame sync (TFSL = 0). The frame sync can also be initiated upon receiving the first bit of data. 0 Transmit frame sync initiated as first bit of data transmits 1 Transmit frame sync is initiated one bit before the data transmits

27.3.11 SSI Receive Configuration Register (SSI_RCR)

The SSI_RCR directs the receive operation of the SSI. A power-on reset clears all SSI_RCR bits. However, an SSI reset does not affect the SSI_RCR bits.

Address: 0xFC0B_C020 (SSI_RCR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	RX EXT	RX BIT0	RFEN1	RFEN0	0	RXDIR	RSHFD	RSCKP	RFSI	RFSL	REFS
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 27-18. SSI Receive Configuration Register (SSI_RCR)

Table 27-11. SSI_RCR Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10 RXEXT	Receive data extension. Allows the SSI to store the received data word in sign-extended form. This bit affects data storage only if the received data is lsb-aligned (RXBIT0 = 1) 0 Sign extension disabled 1 Sign extension enabled
9 RXBIT0	Receive bit 0 (Alignment). Allows SSI to receive the data word at bit position 0 or 15/31 in the receive shift register. The shifting data direction can be msb or lsb first, controlled by the RSHFD bit. 0 msb aligned. Shifting with respect to bit 31 (if word length equals 16, 18, 20, 22 or 24) or bit 15 (if word length equals 8, 10 or 12) of the receive shift register 1 lsb aligned. Shifting with respect to bit 0 of the receive shift register.
8 RFEN1	Receive FIFO enable 1. • When the FIFO is enabled, the FIFO allows eight samples to be received by the SSI (per channel) (a ninth sample can be shifting in) before the SSI_ISR[RDR1] bit is set. • When the FIFO is disabled, SSI_ISR[RDR1] is set when a single sample is received by the SSI. 0 Receive FIFO 1 disabled 1 Receive FIFO 1 enabled
7 RFEN0	Receive FIFO enable 0. Similar description as RFEN1 but pertains to Rx FIFO 0. 0 Receive FIFO 0 disabled 1 Receive FIFO 0 enabled
6	Reserved, must be cleared.
5 RXDIR	Gated clock enable. In synchronous mode, this bit enables gated clock mode. 0 Gated clock mode disabled 1 Gated clock mode enabled
4 RSHFD	Receive shift direction. Controls whether the msb or lsb is received first in a sample. 0 Data received msb first 1 Data received lsb first

Table 27-11. SSI_RCR Field Descriptions (continued)

Field	Description
3 RSCKP	Receive clock polarity. Controls which bit clock edge latches in data for the receive section. 0 Data latched on falling edge of bit clock 1 Data latched on rising edge of bit clock
2 RFSI	Receive frame sync invert. Controls the active state of the frame sync signal for the receive section of SSI. 0 Receive frame sync is active high 1 Receive frame sync is active low
1 RFSL	Receive frame sync length. Controls the length of the frame sync signal generated or recognized for the receive section. The length of a word-long frame sync is the same as the length of the data word selected by SSI_CCR[WL]. 0 Receive frame sync is one word long. 1 Receive frame sync is one bit-clock-period long.
0 REFS	Receive early frame sync. Controls when the frame sync is initiated for the receive section. The frame sync is disabled after one bit for bit length frame sync and after one word for word length frame sync. 0 Receive frame sync initiated as the first bit of data is received. 1 Receive frame sync is initiated one bit before the data is received.

27.3.12 SSI Clock Control Register (SSI_CCR)

The SSI clock control register controls the SSI clock generator, bit and frame sync rates, word length, and number of words per frame for the serial data. The SSI_CCR register controls the receive and transmit sections. Power-on reset clears all SSI_CCR bits, while an SSI reset does not affect these bits.

Address: 0xFC0B_C024 (SSI_CCR)   Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DIV2	PSR	WL				DC					PM							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-19. SSI Clock Control Register (SSI_CCR)

Table 27-12. SSI_CCR Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18 DIV2	Divide-by-2. Controls the divide-by-two divider in series with the rest of the prescalers. 0 Divider bypassed 1 Divider enabled to divide clock by 2
17 PSR	Prescaler range. Controls a fixed divide-by-eight prescaler in series with the variable prescaler. It extends the range of the prescaler for those cases where a slower bit clock is required. 0 Prescaler bypassed 1 Prescaler enabled to divide the clock by 8

Table 27-12. SSI_CCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16–13
WL</td><td>Word length. Controls:
• the length of the data words transferred by the SSI
• the word length divider in the clock generator
• the frame sync pulse length when the FSL bit is cleared
In I^2S master mode, the SSI works with a fixed word length of 32, and the WL bits control the amount of valid data in those 32 bits. Bits per word equal 2 × (WL + 1). Refer to the below table for details of data word lengths supported by the SSI module.
<b>Note:</b> In AC97 mode, if WL is set to any value other than 16 bits, a word length of 20 bits is used.
<table>
<tr><th>WL</th><th>Bits/word</th><th>Supported?</th><th>WL</th><th>Bits/word</th><th>Supported?</th></tr>
<tr><td>0000</td><td>2</td><td>No</td><td>1000</td><td>18</td><td>Yes</td></tr>
<tr><td>0001</td><td>4</td><td>No</td><td>1001</td><td>20</td><td>Yes</td></tr>
<tr><td>0010</td><td>6</td><td>No</td><td>1010</td><td>22</td><td>Yes</td></tr>
<tr><td>0011</td><td>8</td><td>Yes</td><td>1011</td><td>24</td><td>Yes</td></tr>
<tr><td>0100</td><td>10</td><td>Yes</td><td>1100</td><td>26</td><td>No</td></tr>
<tr><td>0101</td><td>12</td><td>Yes</td><td>1101</td><td>28</td><td>No</td></tr>
<tr><td>0110</td><td>14</td><td>No</td><td>1110</td><td>30</td><td>No</td></tr>
<tr><td>0111</td><td>16</td><td>Yes</td><td>1111</td><td>32</td><td>No</td></tr>
</table>
</td></tr>
<tr><td>12–8
DC</td><td>Frame rate divider control. Controls the divide ratio for the programmable frame rate dividers. The divide ratio works on the word clock.
• In normal mode, the ratio determines the word transfer rate. Ranges from 1 to 32.
• In network mode, this field sets the number of words per frame. Ranges from 2 to 32.
In normal mode, a divide ratio of 1 (DC = 00000) provides continuous periodic data word transfer. A bit-length frame sync must be used in this case; otherwise, in word-length mode the frame sync is always asserted.</td></tr>
<tr><td>7–0
PM</td><td>Prescaler modulus select. Controls the prescale divider in the clock generator. This prescaler is used only in internal clock mode to divide the SSI clock. The bit clock output is available at the SSI_BCLK clock pin.
A divide ratio from 1 to 256 (PM = 0x00 to 0xFF) can be selected. Refer to Section 27.4.2.2, “DIV2, PSR and PM Bit Description,” for details regarding settings.</td></tr>
</table>

27.3.13 SSI FIFO Control/Status Register (SSI_FCSR)

See Figure 27-20 for illustration of valid bits in SSI FIFO Control/Status Register and Table 27-13 for description of the bit fields in the register.

0xBASE_2C (SFCSR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	RFCNT1[3:0]				TFCNT1[3:0]				RFWM1[3:0]				TFWM1[3:0]			
RESET	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RFCNT0[3:0]				TFCNT0[3:0]				RFWM0[3:0]				TFWM0[3:0]			
RESET	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Figure 27-20. SSI FIFO Control/Status Register

Table 27-13. SSI FIFO Control/Status Register Field Descriptions

Field	Description
31–28 RFCNT1[3:0]	Receive FIFO Counter1. These bits indicate the number of data words in Receive FIFO 1. Refer to Table 27-14 for details regarding settings for receive FIFO counter bits.

Table 27-14. Receive FIFO Counter Bit Description

Bits	Description
0000	0 data word in receive FIFO
0001	1 data word in receive FIFO
0010	2 data word in receive FIFO
0011	3 data word in receive FIFO
0100	4 data word in receive FIFO
0101	5 data word in receive FIFO
0110	6 data word in receive FIFO
0111	7 data word in receive FIFO
1000	8 data word in receive FIFO
1001	9 data word in receive FIFO
1010	10 data word in receive FIFO
1011	11 data word in receive FIFO
1100	12 data word in receive FIFO
1101	13 data word in receive FIFO
1110	14 data word in receive FIFO
1111	15 data word in receive FIFO

Table 27-13. SSI FIFO Control/Status Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>27–24
TFCNT1[3:0]</td><td>Transmit FIFO Counter1. These bits indicate the number of data words in Transmit FIFO. Refer to Table 27-15 for details regarding settings for transmit FIFO counter bits.

<b>Table 27-15. Transmit FIFO Counter Bit Description</b>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0000</td><td>0 data word in transmit FIFO</td></tr>
<tr><td>0001</td><td>1 data word in transmit FIFO</td></tr>
<tr><td>0010</td><td>2 data word in transmit FIFO</td></tr>
<tr><td>0011</td><td>3 data word in transmit FIFO</td></tr>
<tr><td>0100</td><td>4 data word in transmit FIFO</td></tr>
<tr><td>0101</td><td>5 data word in transmit FIFO</td></tr>
<tr><td>0110</td><td>6 data word in transmit FIFO</td></tr>
<tr><td>0111</td><td>7 data word in transmit FIFO</td></tr>
<tr><td>1000</td><td>8 data word in transmit FIFO</td></tr>
<tr><td>1001</td><td>9 data word in transmit FIFO</td></tr>
<tr><td>1010</td><td>10 data word in transmit FIFO</td></tr>
<tr><td>1011</td><td>11 data word in transmit FIFO</td></tr>
<tr><td>1100</td><td>12 data word in transmit FIFO</td></tr>
<tr><td>1101</td><td>13 data word in transmit FIFO</td></tr>
<tr><td>1110</td><td>14 data word in transmit FIFO</td></tr>
<tr><td>1111</td><td>15 data word in transmit FIFO</td></tr>
</table>
</td></tr>
</table>

Table 27-13. SSI FIFO Control/Status Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>23–20
RFWM1[3:0]</td><td>Receive FIFO Full WaterMark 1. These bits control the threshold at which the RFF1 flag will be set. The RFF1 flag is set whenever the data level in Rx FIFO 1 reaches the selected threshold. Refer to Table 27-16 for details regarding settings for receive FIFO watermark bits.

<b>Table 27-16. Receive FIFO WaterMark Bit Description</b>

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0000</td><td>Reserved</td></tr>
<tr><td>0001</td><td>RFF set when at least one data word have been written to the Receive FIFO
Set when RxFIFO = 1,2.....15 data words</td></tr>
<tr><td>0010</td><td>RFF set when more than or equal to 2 data word have been written to the Receive FIFO.
Set when RxFIFO = 2,3.....15 data words</td></tr>
<tr><td>0011</td><td>RFF set when more than or equal to 3 data word have been written to the Receive FIFO.
Set when RxFIFO = 3,4.....15 data words</td></tr>
<tr><td>0100</td><td>RFF set when more than or equal to 4 data word have been written to the Receive FIFO.
Set when RxFIFO = 4,5.....15 data words</td></tr>
<tr><td>0101</td><td>RFF set when more than or equal to 5 data word have been written to the Receive FIFO.
Set when RxFIFO = 5,6.....15 data words</td></tr>
<tr><td>0110</td><td>RFF set when more than or equal to 6 data word have been written to the Receive.
Set when RxFIFO = 6,7.....15 data words</td></tr>
<tr><td>0111</td><td>RFF set when more than or equal to 7 data word have been written to the Receive FIFO.
Set when RxFIFO = 7,8.....15 data words</td></tr>
<tr><td>1000</td><td>RFF set when more than or equal to 8 data word have been written to the Receive FIFO.
Set when RxFIFO = 8,9.....15 data words</td></tr>
<tr><td>1001</td><td>RFF set when more than or equal to 9 data word have been written to the Receive FIFO.
Set when RxFIFO = 9,10.....15 data words</td></tr>
<tr><td>1010</td><td>RFF set when more than or equal to 10 data word have been written to the Receive FIFO.
Set when RxFIFO = 10,11.....15 data words</td></tr>
<tr><td>1011</td><td>RFF set when more than or equal to 11 data word have been written to the Receive FIFO.
Set when RxFIFO = 11,12.....15 data words</td></tr>
<tr><td>1100</td><td>RFF set when more than or equal to 12 data word have been written to the Receive FIFO.
Set when RxFIFO = 12,13.....15 data words</td></tr>
<tr><td>1101</td><td>RFF set when more than or equal to 13 data word have been written to the Receive FIFO.
Set when RxFIFO = 13,14,15data words</td></tr>
<tr><td>1110</td><td>RFF set when more than or equal to 14 data word have been written to the Receive FIFO.
Set when RxFIFO = 14,15 data words</td></tr>
<tr><td>1111</td><td>RFF set when 15 data word have been written to the Receive FIFO (default).
Set when RxFIFO = 15 data words</td></tr>
</table>
</td></tr>
</table>

Table 27-13. SSI FIFO Control/Status Register Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>19–16
TFWM1[3:0]</td><td>Transmit FIFO Empty WaterMark 1. These bits control the threshold at which the TFE1 flag will be set. The TFE1 flag is set whenever the empty slots in Tx FIFO exceed or are equal to the selected threshold. Refer to Table 27-17 for details regarding settings for transmit FIFO watermark bits.

<b>Table 27-17. Transmit FIFO WaterMark Bit Description</b>
<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>0000</td><td>Reserved</td></tr>
<tr><td>0001</td><td>TFE set when there are more than or equal to 1 empty slots in Transmit FIFO. (default)
Transmit FIFO empty is set when TxFIFO <= 14 data.</td></tr>
<tr><td>0010</td><td>TFE set when there are more than or equal to 2 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 13 data.</td></tr>
<tr><td>0011</td><td>TFE set when there are more than or equal to 3 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 12 data.</td></tr>
<tr><td>0100</td><td>TFE set when there are more than or equal to 4 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 11 data.</td></tr>
<tr><td>0101</td><td>TFE set when there are more than or equal to 5 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 10 data.</td></tr>
<tr><td>0110</td><td>TFE set when there are more than or equal to 6 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 9 data.</td></tr>
<tr><td>0111</td><td>TFE set when there are more than or equal to 7 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 8 data.</td></tr>
<tr><td>1000</td><td>TFE set when there are more than or equal to 8 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 7 data.</td></tr>
<tr><td>1001</td><td>TFE set when there are more than or equal to 9 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 6 data.</td></tr>
<tr><td>1010</td><td>TFE set when there are more than or equal to 10 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 5 data.</td></tr>
<tr><td>1011</td><td>TFE set when there are more than or equal to 11 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 4 data.</td></tr>
<tr><td>1100</td><td>TFE set when there are more than or equal to 12 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 3 data.</td></tr>
<tr><td>1101</td><td>TFE set when there are more than or equal to 13 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 2 data.</td></tr>
<tr><td>1110</td><td>TFE set when there are more than or equal to 14 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO <= 1 data.</td></tr>
<tr><td>1111</td><td>TFE set when there are 15 empty slots in Transmit FIFO.
Transmit FIFO empty is set when TxFIFO = 0 data.</td></tr>
</table>
</td></tr>
</table>

Table 27-13. SSI FIFO Control/Status Register Field Descriptions (continued)

Field	Description
15–12 RFCNT0[3:0]	Receive FIFO Counter 0. These bits indicate the number of data words in Receive FIFO 0. Refer to Table 27-14 for details regarding settings for receive FIFO counter bits.
11–8 TFCNT0[3:0]	Transmit FIFO Counter 0. These bits indicate the number of data words in Transmit FIFO 0. Refer to Table 27-15 for details regarding settings for transmit FIFO counter bits.
7–4 RFWM0[3:0]	Receive FIFO Full WaterMark 0. These bits control the threshold at which the RFF0 flag will be set. The RFF0 flag is set whenever the data level in Rx FIFO 0 reaches the selected threshold. Refer to Table 27-16 for details regarding settings for receive FIFO watermark bits.
3–0 TFWM0[3:0]	Transmit FIFO Empty WaterMark 0. These bits control the threshold at which the TFE0 flag will be set. The TFE0 flag is set whenever the empty slots in Tx FIFO exceed or are equal to the selected threshold. Refer to Table 27-17 for details regarding settings for transmit FIFO watermark bits.

Table 27-18 indicates the status of the Transmit FIFO Empty flag, with different settings of the Transmit FIFO WaterMark bits and varying amounts of data in the Tx FIFO.

Table 27-18. Status of Transmit FIFO Empty Flag

Transmit FIFO Watermark (TFWM)	Number of data in Tx-Fifo														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
4	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
6	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
9	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

27.3.14 SSI AC97 Control Register (SSI_ACR)

SSI_ACR controls various features of the SSI operating in AC97 mode.

Address: 0xFC0B_C038 (SSI_ACR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRDIV						WR	RD	TIF	FV	AC97EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-21. SSI AC97 Control Register (SSI_ACR)

Table 27-19. SSI_ACR Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10–5 FRDIV	Frame rate divider. Controls the frequency of AC97 data transmission/reception. This field is programmed with the number of frames for which the SSI should be idle after operating in one frame. Through these bits, the AC97 frequency of operation, from 48 KHz (000000) to 1 KHz (101111) can be achieved. E.g: 001010 (10 Decimal) equals SSI operates once every 11 frames.
4 WR	Write command. Specifies whether the next frame carries an AC97 write command or not. When this bit is set, the corresponding tag bits (corresponding to command address and command data slots of the next transmit frame) are automatically set. The SSI automatically clears this bit after completing transmission of a frame. 0 Next frame does not have a write command 1 Next frame does have a write command **Note:** Do not set WR and RD at the same time.
3 RD	Read command. Specifies whether the next frame carries an AC97 read command or not. When this bit is set, the corresponding tag bit (corresponding to command address slot of the next transmit frame) is automatically set. The SSI automatically clears this bit after completing transmission of a frame. 0 Next frame does not have a read command 1 Next frame does have a read command **Note:** Do not set WR and RD at the same time.
2 TIF	Tag in FIFO. Controls the destination of the information received in the AC97 tag slot (slot #0). 0 Tag information stored in SSI_ATAG register 1 Tag information stored in Rx FIFO 0
1 FV	Fixed/variable operation. 0 AC97 fixed mode 1 AC97 variable mode
0 AC97EN	AC97 mode enable. Refer to Section 27.4.1.5, “AC97 Mode,” for details of AC97 operation. 0 AC97 mode disabled 1 AC97 mode enabled

27.3.15 SSI AC97 Command Address Register (SSI_ACADD)

SSI_ACADD contains the command address slot information.

Address: 0xFC0B_C03C (SSI_ACADD) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	ACADD																		
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-22. SSI AC97 Command Address Register (SSI_ACADD)

Table 27-20. SSI_ACADD Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18–0 ACADD	AC97 command address. Stores the command address slot information (bit 19 of the slot is sent in accordance with the SSI_ACR[WR and RD] bits). A direct write from the core or the information received in the incoming command address slot can update these bits. If contents of these bits change due to an update, the SSI_ISR[CMDAU] bit is set.

27.3.16 SSI AC97 Command Data Register (SSI_ACDAT)

SSI_ACDAT contains the outgoing command data slot.

Address: 0xFC0B_C040 (SSI_ACDAT) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	ACDAT																			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-23. SSI AC97 Command Data Register (SSI_ACDAT)

Table 27-21. SSI_ACDAT Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–0 ACDAT	AC97 command data. The outgoing command data slot carries the information contained in these bits. A direct write from the core or the information received in the incoming command data slot can update these bits. If the contents of these bits change due to an update, the SSI_ISR[CMDDU] bit is set. During an AC97 read command, 0x0_0000 in time slot #2.

27.3.17 SSI AC97 Tag Register (SSI_ATAG)

Address: 0xFC0B_C044 (SSI_ATAG) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ATAG															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-24. SSI AC97 Tag Register (SSI_ATAG)

Table 27-22. SSI_ATAG Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 ATAG	AC97 tag. Writing to this register sets the value of the Tx tag (in AC97 fixed mode). On a read, the processor gets the last Rx tag value received. It is updated at the start of each received frame. The contents of this register also generate the transmit tag in AC97 variable mode. When the received tag value changes, the SSI_ISR[RXT] bit is set, if enabled. If the SSI_ACR[TIF] bit is set, the TAG value is also stored in Rx FIFO. **Note:** Bits 1–0 convey the codec-ID. Because only primary codecs are supported, these bits must be cleared.

27.3.18 SSI Transmit Time Slot Mask Register (SSI_TMASK)

This register controls the time slots that the SSI transmits data in network mode.

Address: 0xFC0B_C048 (SSI_TMASK) Access: User 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
R	TMASK																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-25. SSI Transmit Time Slot Mask Register (SSI_TMASK)

Table 27-23. SSI_TMASK Field Descriptions

Field	Description
31–0 TMASK	Transmit mask. Indicates which transmit time slot has been masked in the current frame. Each bit corresponds to the respective time slot in the frame. If a change is made to the register contents, the transmission pattern is updated from the next time slot. Transmit mask bits should not be used in I^2S slave mode. 0 Valid time slot 1 Time slot masked (no data transmitted in this time slot)

27.3.19 SSI Receive Time Slot Mask Register (SSI_RMASK)

This register controls the time slots that the SSI receives data in network mode.

Address: 0xFC0B_C04C (SSI_RMASK) Access: User 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
R W	RMASK																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-26. SSI Receive Time Slot Mask Register (SSI_RMASK)

Table 27-24. SSI_RMASK Field Descriptions

Field	Description
31–0 RMASK	Receive mask. Indicates which received time slot has been masked in the current frame. Each bit corresponds to the respective time slot in the frame. If a change is made to the register contents, the reception pattern is updated from the next time slot. Receive mask bits should not be used in I^{2}S slave mode. 0 Valid time slot 1 Time slot masked (no data received in this time slot)

27.4 Functional Description

27.4.1 Detailed Operating Mode Descriptions

The following sections describe in detail the main operating modes of the SSI module: normal, network, gated clock, I^{2}S, and AC97.

27.4.1.1 Normal Mode

Normal mode is the simplest mode of the SSI. It transfers data in one time slot per frame. A time slot is a unit of data and the WL bits define the number of bits in a time slot. In continuous clock mode, a frame sync occurs at the beginning of each frame. The following factors determine the length of the frame:

- Period of the serial bit clock (DIV2, PSR, PM bits for internal clock or the frequency of the external clock on the SSI_BCLK port)
- Number of bits per time slot (WL bits)
- Number of time slots per frame (DC bits)

If normal mode is configured with more than one time slot per frame, data transfers only in the first time slot of the frame. No data transfers in subsequent time slots. In normal mode, DC values corresponding to more than a single time slot in a frame only result in lengthening the frame.

27.4.1.1.1 Normal Mode Transmit

Conditions for data transmission from the SSI in normal mode are:

1. SSI enabled (SSI_CR[SSI_EN] = 1)
2. Enable FIFO and configure transmit and receive watermark if the FIFO is used.

3. Write data to transmit data register (SSI_TX0)
4. Transmitter enabled (TE = 1)
5. Frame sync active (for continuous clock case)
6. Bit clock begins (for gated clock case)

When the above conditions occur in normal mode, the next data word transfers into the transmit shift register (TXSR) from the transmit data register 0 (SSI_TX0) or from the transmit FIFO 0 register, if enabled. The new data word transmits immediately.

If transmit FIFO 0 is not enabled and the transmit data register empty (TDE0) bit is set, a transmit interrupt 0 occurs if the TIE and SSI_IER[TDE0] bits are set.

If transmit FIFO 0 is enabled and the transmit FIFO empty (TFE0) bit is set, transmit interrupt 0 occurs if the TIE and SSI_IER[TFE0] bits are set. If transmit FIFO 0 is enabled and filled with data, eight data words can be transferred before the core must write new data to the SSI_TX0 register.

The SSI_TXD port is disabled except during the data transmission period. For a continuous clock, the optional frame sync output and clock outputs are not disabled, even if receiver and transmitter are disabled.

27.4.1.1.2 Normal Mode Receive

Conditions for data reception from the SSI are:

1. SSI enabled (SSI_CR[SSI_EN] = 1)
2. Enable receive FIFO (optional)
3. Receiver enabled (RE = 1)
4. Frame sync active (for continuous clock case)
5. Bit clock begins (for gated clock case)

With the above conditions in normal mode with a continuous clock, each time the frame sync signal is generated (or detected), a data word is clocked in. With the above conditions and a gated clock, each time the clock begins, a data word is clocked in.

If receive FIFO 0 is not enabled, the received data word is transferred from the receive shift register (RXSR) to the receive data register 0 (SSI_RX0), and the RDR0 flag is set. Receive interrupt 0 occurs if the RIE and SSI_IER[RDR0] bits are set.

If receive FIFO 0 is enabled, the received data word is transferred to the receive FIFO 0. The RFF0 flag is set if the receive data register (SSI_RX0) is full and receive FIFO 0 reaches the selected threshold. Receive interrupt 0 occurs if RIE and SSI_IER[RFF0] bits are set.

The core has to read the data from the SSI_RX0 register before a new data word is transferred from the RXSR; otherwise, receive overrun error 0 (ROE0) bit is set. If receive FIFO 0 is enabled, the ROE0 bit is set when the receive FIFO 0 data level reaches the selected threshold and a new data word is ready to transfer to the receive FIFO 0.

Figure 27-27 shows transmitter and receiver timing for an 8-bit word with two words per time slot in normal mode and continuous clock with a late word length frame sync. The Tx data register is loaded with the data to be transmitted. On arrival of the frame sync, this data is transferred to the transmit shift register and transmitted on the SSI_TXD output. Simultaneously, the receive shift register shifts in the received data available on the SSI_RXD input. At the end of the time slot, this data is transferred to the Rx data register.

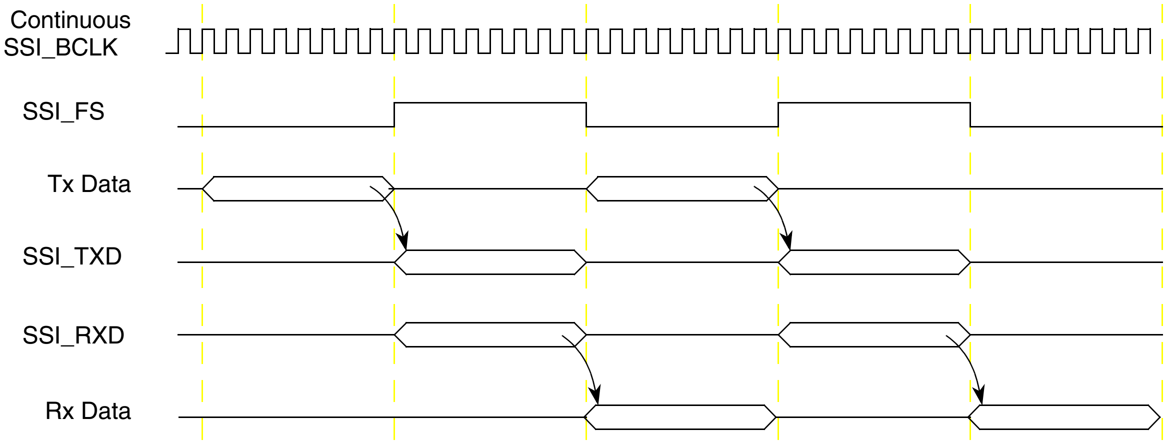


Figure 27-27. Normal Mode Timing - Continuous Clock

Figure 27-28 shows a similar case for internal (SSI generates clock) gated clock mode, and Figure 27-29 shows a case for external (SSI receives clock) gated clock mode.

NOTE

A pull-down resistor is required in gated clock mode, because the clock port is disabled between transmissions.

The Tx data register is loaded with the data to be transmitted. On arrival of the clock, this data transfers to the transmit shift register and transmits on the SSI_TXD output. Simultaneously, the receive shift register shifts in the received data available on the SSI_RXD input, and at the end of the time slot, this data transfers to the Rx data register. In internal gated clock mode, the Tx data line and clock output port are tri-stated at the end of transmission of the last bit (at the completion of the complete clock cycle). Whereas, in external gated clock mode, the Tx data line is tri-stated at the last inactive edge of the incoming bit clock (during the last bit in a data word).

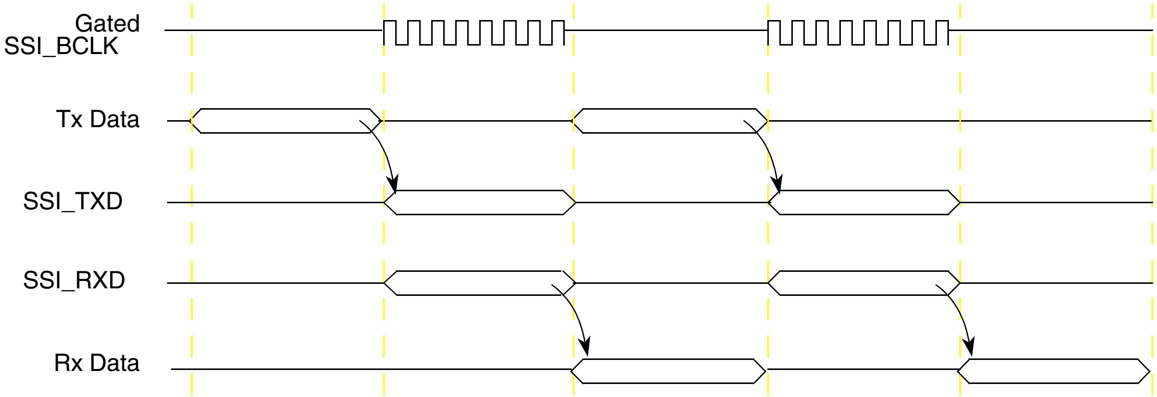


Figure 27-28. Normal Mode Timing - Internal Gated Clock

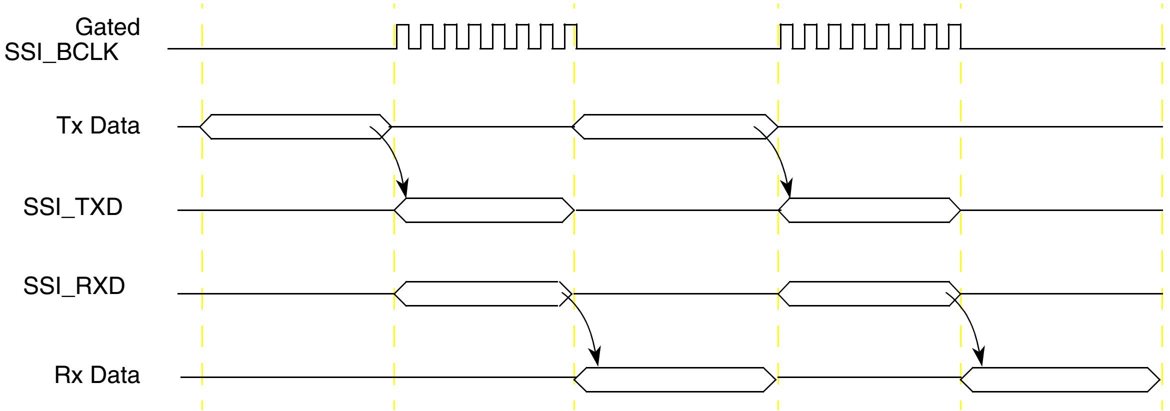


Figure 27-29. Normal Mode Timing - External Gated Clock

27.4.1.2 Network Mode

Network mode creates a time division multiplexed (TDM) network, such as a TDM codec network or a network of DSPs. In continuous clock mode, a frame sync occurs at the beginning of each frame. In this mode, the frame is divided into more than one time slot. During each time slot, one data word can be transferred (rather than in the frame sync time slot as in normal mode). Each time slot is then assigned to an appropriate codec or DSP on the network. The processor can be a master device that controls its own private network or a slave device connected to an existing TDM network and occupies a few time slots.

The frame rate dividers, controlled by the DC bits, select two to thirty-two time slots per frame. The length of the frame is determined by:

- The period of the serial bit clock (PSR, PM bits for internal clock, or the frequency of the external clock on the SSI_BCLK pin)
- The number of bits per sample (WL bits)
- The number of time slots per frame (DC bits)

In network mode, data can be transmitted in any time slot. The distinction of network mode is each time slot is identified with respect to the frame sync (data word time). This time slot identification allows the option of transmitting data during the time slot by writing to the SSI_TX registers or ignoring the time slot as determined by the SSI_TMASK register bits. The receiver is treated in the same manner and received data is only transferred to the receive data register/FIFO if the corresponding time slot is enabled through SSI_RMASK.

By using the SSI_TMASK and SSI_RMASK registers, software only has to service the SSI during valid time slots. This eliminates any overhead associated with unused time slots. Refer to Section 27.3.18, “SSI Transmit Time Slot Mask Register (SSI_TMASK),” and Section 27.3.19, “SSI Receive Time Slot Mask Register (SSI_RMASK),” for more information on the SSI_TMASK and SSI_RMASK registers.

In two channel mode (SSI_CR[TCH] = 1), the second set of transmit and receive FIFOs and data registers create two separate channels (for example, left and right channels for a stereo codec). These channels are completely independent with their own set of interrupts and DMA requests identical to the ones available for the default channel. In this mode, data is transmitted/received in enabled time slots alternately from/to FIFO 0 and FIFO 1, starting from FIFO 0. The first data word is taken from FIFO 0 and transmitted in the first enabled time slot and subsequently, data is loaded from FIFO 1 and FIFO 0 alternately and transmitted. Similarly, the first received data is sent to FIFO 0 and subsequent data is sent to FIFO 1 and FIFO 0 alternately. Time slots are selected through the transmit and receive time slot mask registers (SSI_TMASK and SSI_RMASK).

27.4.1.2.1 Network Mode Transmit

The transmit portion of SSI is enabled when the SSI_CR[SSI_EN and TE] bits are set. However, for continuous clock when the TE bit is set, the transmitter is enabled only after detection of a new frame sync (transmission starts from the next frame boundary).

Normal start-up sequence for transmission:

- Write the data to be transmitted to the SSI_TX register. This clears the TDE flag.
- Set the SSI_CR[TE] bit to enable the transmitter on the next word boundary (for continuous clock).
- Enable transmit interrupts.

Alternately, the user may decide not to transmit in a time slot by writing to the SSI_TMASK. The TDE flag is not cleared, but the SSI_TXD port remains disabled during the time slot. When the frame sync is detected or generated (continuous clock), the first enabled data word is transferred from the SSI_TX register to the TXSR and is shifted out (transmitted). When the SSI_TX register is empty, the TDE bit is set, which causes a transmitter interrupt (if the FIFO is disabled) to be sent if the TIE bit is set. Software can poll the TDE bit or use interrupts to reload the SSI_TX register with new data for the next time slot. Failing to reload the SSI_TX register before the TXSR is finished shifting (empty) causes a transmitter underrun error (the TUE bit is set). If the FIFO is enabled, the TFE flag is set in accordance with the watermark setting and this flag causes a transmitter interrupt to occur.

Clearing the TE bit disables the transmitter after completion of transmission of the current frame. Setting the TE bit enables transmission from the next frame. During that time the SSI_TXD port is disabled. The TE bit should be cleared after the TDE bit is set to ensure that all pending data is transmitted.

To summarize, the network mode transmitter generates interrupts every enabled time slot and requires the processor to respond to each enabled time slot. These responses may be:

- Write data in data register to enable transmission in the next time slot.
- Configure the time slot register to disable transmission in the next time slot (unless the time slot is already masked by the SSI_TMASK register bit).
- Do nothing—transmit underrun occurs at the beginning of the next time slot and the previous data is re-transmitted.

In two channel operation, both channels (data registers, FIFOs, interrupts, and DMA requests) operate in the same manner, as described above. The only difference is interrupts related to the second channel are generated only if this mode of operation is selected (TDE1 is low by default).

27.4.1.2.2 Network Mode Receive

The receiver portion of the SSI is enabled when both the SSI_CR[SSI_EN and RE] bits are set. However, the receive enable only takes place during that time slot if RE is enabled before the second to last bit of the word. If the RE bit is cleared, the receiver is disabled at the end of the current frame. The SSI module is capable of finding the start of the next frame automatically. When the word is completely received, it is transferred to the SSI_RX register, which sets the RDR bit. This causes a receive interrupt to occur if the the RIE bit is set. The second data word (second time slot in the frame) begins shifting in immediately after the transfer of the first data word to the SSI_RX register. The processor has to read the data from the receive data register (which clears RDR) before the second data word is completely received (ready to transfer to RX data register) or a receive overrun error occurs (the ROE bit is set).

An interrupt can occur after the reception of each enabled data word or the user can poll the RDR flag. The processor response can be:

- Read RX and use the data.
- Read RX and ignore the data.
- Do nothing—the receiver overrun exception occurs at the end of the current time slot.

NOTE

For a continuous clock, the optional frame sync output and clock output signals are not affected, even if transmitter or receiver is disabled. TE and RE do not disable the bit clock or the frame sync generation. To disable the bit clock and the frame sync generation, the SSI_CR[SSI_EN] bit can be cleared or the port control logic external to the SSI (e.g. GPIO) can be reconfigured.

In two channel operation, both the channels (data registers, FIFOs, interrupts, and DMA requests) operate in the same manner as described above. The only difference is second channel interrupts are generated only in this mode of operation.

Figure 27-30 shows the transmitter and receiver timing for an 8-bit word with continuous clock, FIFO disabled, three words per frame sync in network mode.

NOTE

The transmitter repeats the value 0x5E because of an underrun condition.

For the transmit section, the SSI_TMASK value is updated in the last time slot of frame 1 to mask the first two time slots (0x3). This value takes effect at the next time slot and, consequently, the next frame transmits data in the third time slot only.

For the receive section, data received on the SSI_RXD pin is transferred to the SSI_RX register at the end of each time slot. If the FIFO is disabled, RDR flag sets and causes a receiver interrupt if the RE, RIE, and SSI_IER[RDR] bits are set. If the FIFO is enabled, the RFF flag generates interrupts (this flag is set in accordance with the watermark settings). In this example all time slots are enabled. The receive data ready flag is set after reception of the first data (0x55). Because the flag is not cleared (Rx data register is not read), the receive overrun error (ROE) flag is set on reception of the next data (0x5E). The ROE flag is cleared by reading the SSI status register followed by reading the Rx data register.

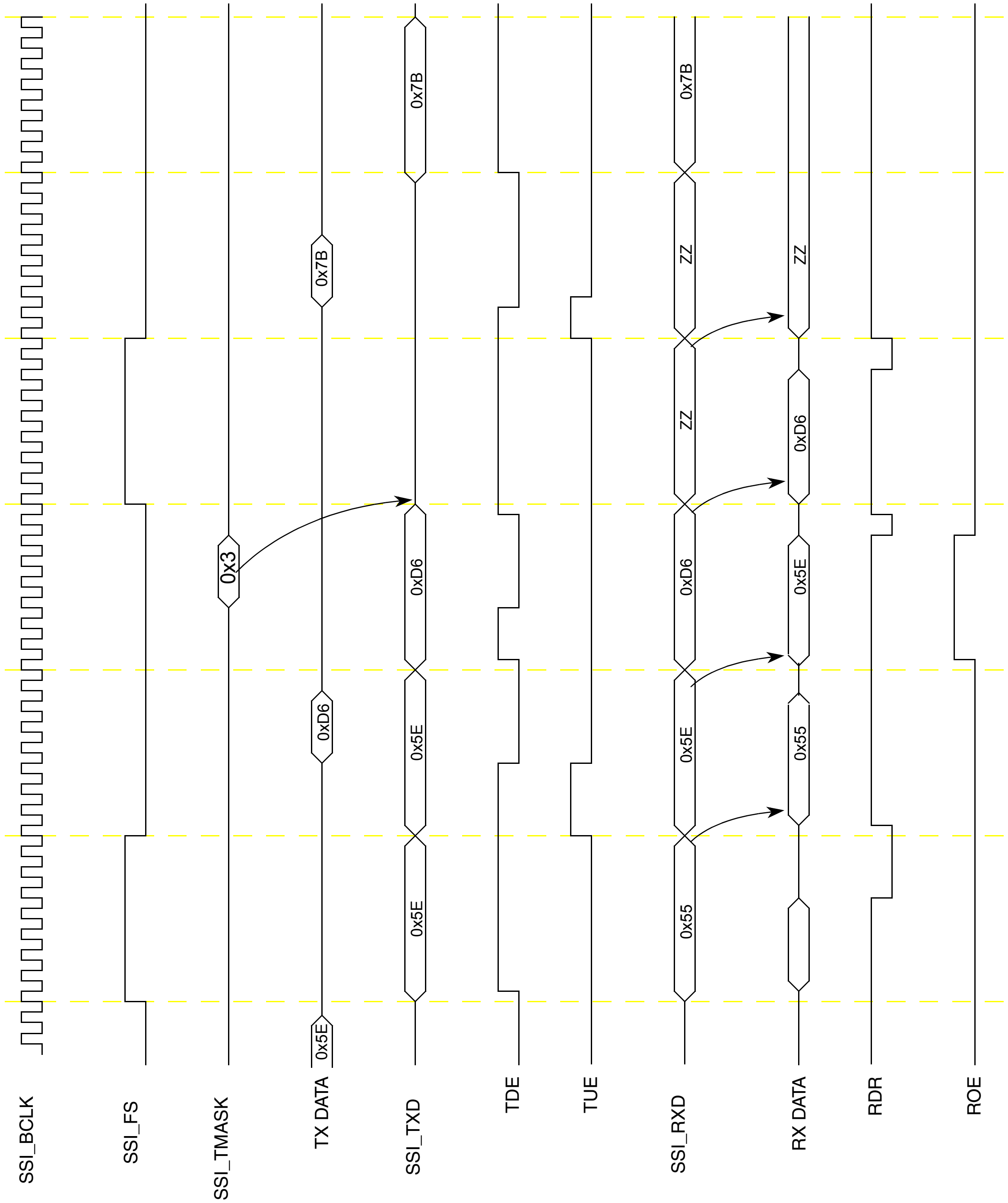


Figure 27-30. Network Mode Timing - Continuous Clock

27.4.1.3 Gated Clock Mode

Gated clock mode often connects to SPI-type interfaces on microcontroller units (MCUs) or external peripheral devices. In gated clock mode, presence of the clock indicates that valid data is on the SSI_TXD or SSI_RXD signals. For this reason, no frame sync is needed in this mode. After transmission of data completes, the clock is pulled to the inactive state. Gated clocks are allowed for the transmit and receive sections with internal or external clock and in normal mode. Gated clocks are not allowed in network mode. Refer to Table 27-3 for SSI configuration for gated mode operation.

The clock operates when the TE bit and/or the RE bit are appropriately enabled. For an internally generated clock, all internal bit clocks, word clocks, and frame clocks continue to operate. When a valid time slot occurs (such as the first time slot in normal mode), the internal bit clock is enabled onto the clock port. This allows data to be transferred out in periodic intervals in gated clock mode. With an external clock, the SSI module waits for a clock signal to be received. After the clock begins, valid data is shifted in. Care should be taken to clear all DC bits when the module is used in gated mode.

For gated clock operated in external clock mode, proper clock signalling must apply to SSI_BCLK for it to function properly. If the SSI uses rising edge transition to clock data (TSCKP = 0) and falling edge transition to latch data (RSCKP = 0), the clock must be in an active low state when idle. If the SSI uses falling edge transition to clock data (TSCKP = 1) and rising edge transition to latch data (RSCKP = 1), the clock must be in a active high state when idle. The following diagrams illustrate the different edge clocking/latching.

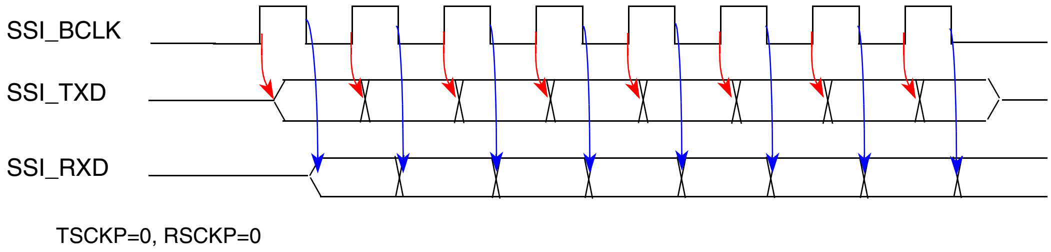


Figure 27-31. Internal Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching

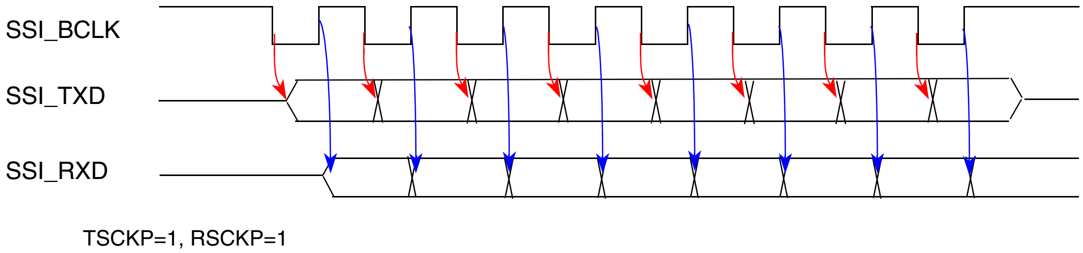


Figure 27-32. Internal Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

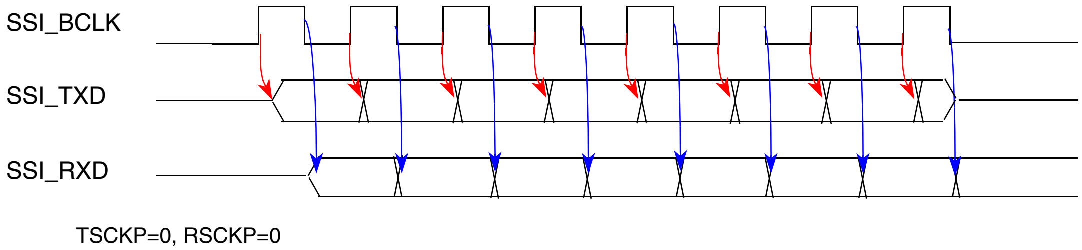


Figure 27-33. External Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching

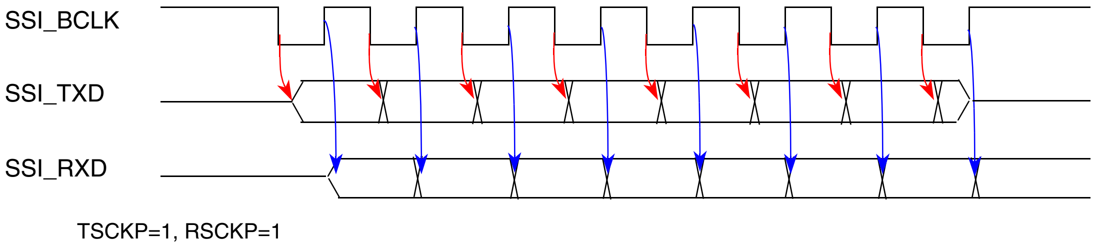


Figure 27-34. External Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

NOTE

The bit clock signals must not have timing glitches. If a single glitch occurs, all ensuing transfers are out of synchronization.

NOTE

In external gated mode, even though the transmit data line is tri-stated at the last non-active edge of the bit clock, the round trip delay should sufficiently take care of hold time requirements at the external receiver.

27.4.1.4 I²S Mode

The SSI is compliant to I²S bus specification from Philips Semiconductors (February 1986, Revised June 5, 1996). Figure 27-35 depicts basic I²S protocol timing.

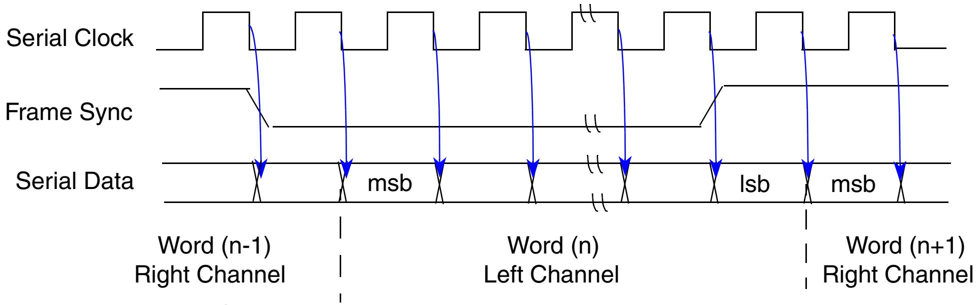


Figure 27-35. I²S Mode Timing - Serial Clock, Frame Sync and Serial Data

I²S mode can be selected by the SSI_CR[I2S] bits as follows:

Table 27-25. I²S Mode Selection

SSI_CR[I2S]	Mode
00	Normal mode
01	I²S master mode
10	I²S slave mode
11	Normal mode

In normal (non-I²S) mode operation, no register bits are forced to any particular state internally, and the user can program the SSI to work in any operating condition.

When I^{2}S modes are entered (SSI_CR[I2S] = 01 or 10), these settings are recommended:

- Synchonous mode (SSI_CR[SYN] = 1)
- Tx shift direction: msb transmitted first (SSI_TCR[TSHFD] = 0)
- Rx shift direction: msb received first (SSI_RCR[RSHFD] = 0)
- Tx data clocked at falling edge of the clock (SSI_TCR[TSCKP] = 1)
- Rx data latched at rising edge of the clock (SSI_RCR[RSCKP] = 1)
- Tx frame sync active low (SSI_TCR[TFSI] = 1)
- Rx frame sync active low (SSI_RCR[RFSI] = 1)
- Tx frame sync initiated one bit before data is transmitted (SSI_TCR[TEFS] = 1)
- Rx frame sync initiated one bit before data is received (SSI_RCR[REFS] = 1)

27.4.1.4.1 I^{2}S Master Mode

In I^{2}S master mode (SSI_CR[I2S] = 01), these additional settings are recommended:

- Internal generated bit clock (SSI_TCR[TXDIR] = 1)
- Internal generated frame sync (SSI_TCR[TFDIR] = 1)

The processor automatically performs these settings when in I^{2}S master mode:

- Network mode is selected (SSI_CR[NET] = 1)
- Tx frame sync length set to one-word-long-frame (SSI_TCR[TFSL] = 0)
- Rx frame sync length set to one-word-long-frame (SSI_RCR[RFSL] = 0)
- Tx shifting w.r.t. bit 0 of TXSR (SSI_TCR[TXBIT0] = 1)
- Rx shifting w.r.t. bit 0 of RXSR (SSI_RCR[RXBIT0] = 1)

Set the SSI_CCR[PM, PSR, DIV2, WL, DC] control bits to configure the bit clock and frame sync.

The word length is fixed to 32 in I^{2}S master mode, and the WL bits determine the number of bits that contain valid data (out of the 32 transmitted/received bits in each channel). The fixing of word duration as 32 simplifies the relation between oversampling clock (SSI_MCLK) and the frame sync (SSI_MCLK becomes an integer multiple of frame sync). The period of the oversampling clock must be at least 4x the internal bus clock period.

27.4.1.4.2 I^{2}S Slave Mode

In I^{2}S slave mode (SSI_CR[I2S] = 10), the following additional settings are recommended:

- External generated bit clock (SSI_TCR[TXDIR] = 0)
- External generated frame sync (SSI_TCR[TFDIR] = 0)

The following settings are done automatically by the processor when in I^{2}S slave mode:

- Normal mode is selected (SSI_CR[NET] = 0)
- Tx frame sync length set to one-bit-long-frame (SSI_TCR[TFSL] = 1)
- Rx frame sync length set to one-bit-long-frame (SSI_RCR[RFSL] = 1)
- Tx shifting w.r.t. bit 0 of TXSR (SSI_TCR[TXBIT0] = 1)
- Rx shifting w.r.t. bit 0 of RXSR (SSI_RCR[RXBIT0] = 1)

Set the SSI_CCR[WL, DC] bits to configure the data transmission.

The word length is variable in I^{2}S slave mode and the WL bits determine the number of bits that contain valid data. The actual word length is determined by the external codec. The external I^{2}S master sends a frame sync according to the I^{2}S protocol (early, word wide, and active low). The SSI internally operates so each frame sync transition is the start of a new frame (the WL bits determine the number of bits to be transmitted/received). After one data word has been transferred, the SSI waits for the next frame sync transition to start operation in the next time slot. Transmit and receive mask bits should not be used in I^{2}S slave mode.

27.4.1.5 AC97 Mode

In AC97 mode, SSI transmits a 16-bit tag slot at the start of a frame and the rest of the slots (in that frame) are all 20-bits wide. The same sequence is followed while receiving data. Refer to the AC97 specification for details regarding transmit and receive sequences and data formats.

NOTE

Since the SSI has only one RxDATA pin, only one codec is supported. Secondary codecs are not supported.

When AC97 mode is enabled, the hardware internally overrides the following settings. The programmed register values are not changed by entering AC97 mode, but they no longer apply to the module's operation. Writing to the programmed register fields updates their values. These updates can be seen by reading back the register fields. However, these settings do not take effect until AC97 mode is turned off.

The register bits within the bracket are equivalent settings.

- Synchronous mode is entered (SSI_CR[SYN] = 1)
- Network mode is selected (SSI_CR[NET] = 1)
- Tx shift direction is msb transmitted first (SSI_TCR[TSHFD] = 0)
- Rx shift direction is msb received first (SSI_RCR[RSHFD] = 0)
- Tx data is clocked at rising edge of the clock (SSI_TCR[TSCKP] = 0)
- Rx data is latched at falling edge of the clock (SSI_RCR[RSCKP] = 0)
- Tx frame sync is active high (SSI_TCR[TFSI] = 0)
- Rx frame sync is active high (SSI_RCR[RFSI] = 0)
- Tx frame sync length is one-word-long-frame (SSI_TCR[TFSL] = 0)
- Rx frame sync length is one-word-long-frame (SSI_RCR[RFSL] = 0)
- Tx frame sync initiated one bit before data is transmitted (SSI_TCR[TEFS] = 1)

- Rx frame sync initiated one bit before data is received (SSI_RCR[REFS] = 1)
- Tx shifting w.r.t. bit 0 of TXSR (SSI_TCR[TXBIT0] = 1)
- Rx shifting w.r.t. bit 0 of RXSR (SSI_RCR[RXBIT0] = 1)
- Tx FIFO is enabled (SSI_TCR[TFEN0] = 1)
- Rx FIFO is enabled (SSI_RCR[RFEN0] = 1)
- Internally-generated frame sync (SSI_TCR[TFDIR] = 1)
- Externally-generated bit clock (SSI_TCR[TXDIR] = 0)

Any alteration of these bits does not affect the operational conditions of the SSI unless AC97 mode is deselected. Hence, the only control bits that need to be set to configure the data transmission/reception are the SSI_CCR[WL, DC] bits. In AC97 mode, the WL bits can only legally take the values corresponding to 16-bit (truncated data) or 20-bit time slots. If the WL bits are set to select 16-bit time slots, while receiving, the SSI pads the data (four least significant bits) with 0s, and while receiving, the SSI stores only the 16 most significant bits in the Rx FIFO.

The following sequence should be followed for programming the SSI to work in AC97 mode:

1. Program the SSI_CCR[WL] bits to a value corresponding to 16 or 20 bits. The WL bit setting is only for the data portion of the AC97 frame (slots #3 through #12). The tag slot (slot #0) is always 16-bits wide and the command address and command data slots (slots #1 and #2) are always 20 bits wide.
2. Select the number of time slots through the SSI_CCR[DC] bits. For AC97 operation, the DC bits should be set to a value of 0xC, resulting in 13 time slots per frame.
3. Write data to be transmitted in Tx FIFO 0 (through Tx data register 0)
4. Program the SSI_ACR[FV, TIF, RD, WR and FRDIV] bits
5. Update the contents of SSI_ACADD, SSI_ACDAT and SSI_ATAG (for fixed mode only) registers
6. Enable AC97 mode (SSI_ACR[AC97EN] bit)

After the SSI starts transmitting and receiving data after being configured in AC97 mode, the processor needs to service the interrupts when they are raised (updates to command address/data or tag registers, reading of received data, and writing more data for transmission). Further details regarding fixed and variable mode implementation appear in the following sections.

While using AC97 in two-channel mode (TCH = 1), it is recommended that the received tag is not stored in the Rx FIFO (TIF = 0). If you need to update the SSI_ATAG register and also issue a RD/WR command (in a single frame), it is recommended that the SSI_ATAG register is updated prior to issuing a RD/WR command.

27.4.1.5.1 AC97 Fixed Mode (SSI_ACR[FV]=0)

In fixed mode of operation, SSI transmits in accordance with the frame rate divider bits that decide the number of frames for which the SSI should be idle, after operating for one frame. The following shows the slot assignments in a valid transmit frame:

- Slot 0: The tag value (written by the user program)
- Slot 1: If RD/WR command, command address

- Slot 2: If WR command, command data
- Slot 3–12: Transmit FIFO data, depending on the valid slots indicated by the TAG value

While receiving, bit 15 of the received tag slot (slot 0) is checked to see if the codec is ready. If this bit is set, the frame is received. The received tag provides the information about slots containing valid data. If the corresponding tag bit is valid, the command address (slot 1) and command data (slot 2) values are stored in the corresponding registers. The received data (slot 3–12) is then stored in the receive FIFO (for valid slots).

27.4.1.5.2 AC97 Variable Mode (SSI_ACR[FV]=1)

In variable mode, the transmit slots that should contain data in the current frame are determined by the SLOTREQ bits received in slot 1 of the previous frame. While receiving, if the codec is ready, the frame is received and the SLOTREQ bits are stored for scheduling transmission in the next frame.

The SACCST, SACCEN and SACCDIS registers help determine which transmit slots are active. This information is used to ensure that SSI does not transmit data for powered-down/inactive channels.

27.4.2 SSI Clocking

The SSI uses the following clocks:

- SSI_CLOCK — This is the internal clock that drives the SSI’s clock generation logic, which can be a fraction of the internal core clock (f_{sys}) or the clock input on the SSI_CLKIN pin. The CCM’s MISCCR register can select either of these sources. Having this choice allows the user to operate the SSI module at frequencies that would not be achievable if standard internal core clock frequencies are used. This is also the output master clock (SSI_MCLK) when in master mode.
- Bit clock — Serially clocks the data bits in and out of the SSI port. This clock is generated internally or taken from external clock source (through SSI_BCLK).
- Word clock — Counts the number of data bits per word (8, 10, 12, 16, 18, 20, 22 or 24 bits). This clock is generated internally from the bit clock.
- Frame clock (frame sync) — Counts the number of words in a frame. This signal can be generated internally from the bit clock or taken from external source (from SSI_FS).
- Master clock — In master mode, this is an integer multiple of frame clock. It is used in cases when SSI has to provide a clock to the connected devices.

Take care to ensure that the bit clock frequency (internally generated or sourced from an external device) is never greater than 1/5 of the internal bus frequency ($f_{sys/2}$).

In normal mode, the bit clock, used to serially clock the data, is visible on the serial clock (SSI_BCLK) port. The word clock is an internal clock that determines when transmission of an 8, 10, 12, 16, 18, 20, 22, or 24-bit word has completed. The word clock then clocks the frame clock, which counts the number of words in the frame. The frame clock can be viewed on the SSI_FS frame sync port because a frame sync generates after the correct number of words in the frame have passed. In master mode, the SSI_MCLK signal is the serial master clock if enabled by the SSI_CR[MCE] bit. This serial master clock is an oversampling clock of the frame sync clock (SSI_FS). In this mode, the word length (WL), prescaler range

(PSR), prescaler modulus (PM), and frame rate (DC) selects the ratio of SSI_MCLK to sampling clock, SSI_FS. In I^{2}S mode, the oversampling clock is available on this port if the SSI_CR[MCE] bit is set.

Figure 27-36 shows the relationship between the clocks and the dividers. The bit clock can be received from an SSI clock port or generated from the internal clock (SSI_CLOCK) through a divider, as shown in Figure 27-37.

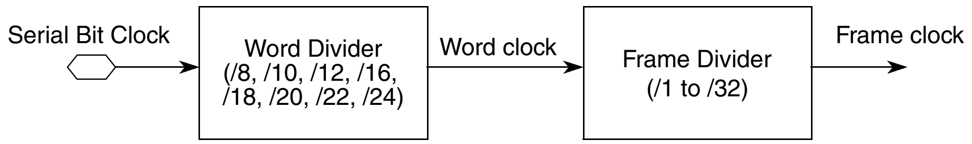


Figure 27-36. SSI Clocking

27.4.2.1 SSI Clock and Frame Sync Generation

Data clock and frame sync signals can be generated internally or obtained from external sources. If internally generated, the SSI clock generator derives bit clock and frame sync signals from the SSI_CLOCK. The SSI clock generator consists of a selectable, fixed prescaler and a programmable prescaler for bit rate clock generation. A programmable frame rate divider and a word length divider are used for frame rate sync signal generation.

Figure 27-37 shows a block diagram of the clock generator for the transmit section. The serial bit clock can be internal or external, depending on the transmit direction (SSI_TCR[TXDIR]) bit.

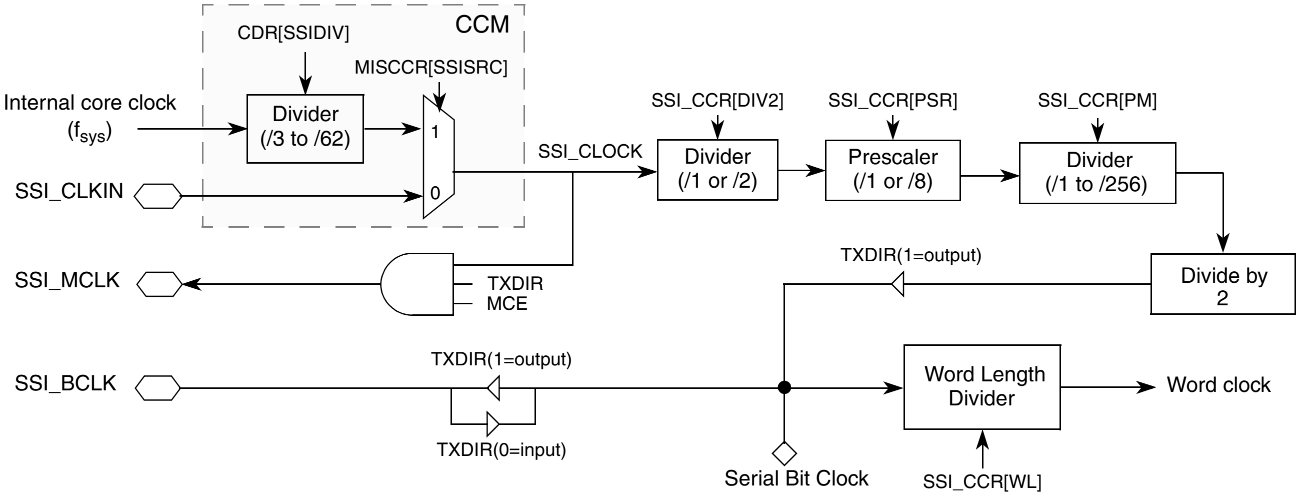


Figure 27-37. SSI Transmit Clock Generator Block Diagram

Figure 27-38 shows the frame sync generator block for the transmit section. When internally generated, receive and transmit frame sync generate from the word clock and are defined by the frame rate divider (DC) bits and the word length (WL) bits of the SSI_CCR.

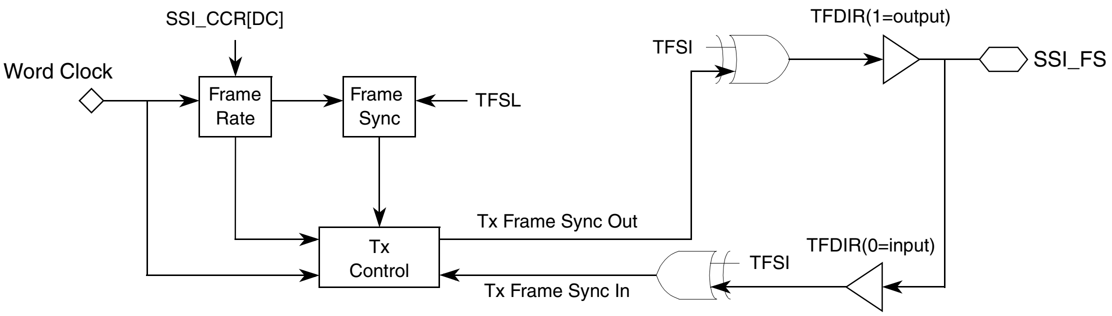


Figure 27-38. SSI Transmit Frame Sync Generator Block Diagram

27.4.2.2 DIV2, PSR and PM Bit Description

The bit clock frequency can be calculated from the SSI serial system clock (SSI_CLOCK), using Equation 27-1.

NOTE

> You must ensure that the bit-clock frequency is at most one-fifth the internal bus frequency ($f_{sys/2}$). The oversampling clock frequency can go up to internal bus frequency. Bits DIV2, PSR, and PM must not be cleared at the same time.

$$f_{INT\_BIT\_CLK} = \frac{\text{SSI serial system clock}}{(DIV2 + 1) \times (7 \times PSR + 1) \times (PM + 1) \times 2} \qquad \textit{Eqn. 27-1}$$

From this, the frame clock frequency can be calculated:

$$f_{FS\_CLK} = \frac{f_{INT\_BIT\_CLK}}{(DC + 1) \times (2 \times (WL + 1))} \qquad \textit{Eqn. 27-2}$$

For example, if the SSI working clock is 19.2 MHz, in 8-bit word normal mode with DC = 1, PM = 0x4A (74), PSR = 0, DIV2 = 1, a bit clock rate of 64 kHz is generated. Because the 8-bit word rate equals two, sampling rate (or frame sync rate) would then be 64/(2×8) = 4 kHz.

In the next example, SSI_CLOCK is 12 MHz. A 16-bit word network mode with DC = 1, PM = 1, the PSR = 0, DIV2 = 1, a bit clock rate of 12/[14×2] = 1.5 MHz is generated. Because the 16-bit word rate equals two, sampling rate (or frame sync rate) would be 1.5/(2×16) = 46.875 kHz.

Table 27-26 shows the example of programming PSR and PM bits to generate different bit clock (SSI_BCLK) frequencies. The SSI_CLKIN signal is used in this example (MISCCR[SSISRC] = 0) because when operating the processor at the typical 266 MHz frequency, the SSI module is not able to accurately produce standard bit and sample rates.

Table 27-26. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2

SSI_CLKIN freq (MHz) (SSI_MCLK)	SSI_CCR					Bit Clk (kHz) SSI_BCLK	Frame rate (kHz)
	DIV2	PSR	PM	WL	DC		
12.288	0	0	23	3	3	256	8
12.288	0	0	11	3	3	512	16

Table 27-26. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2 (continued)

SSI_CLKIN freq (MHz) (SSI_MCLK)	SSI_CCR					Bit Clk (kHz) SSI_BCLK	Frame rate (kHz)
	DIV2	PSR	PM	WL	DC		
12.288	0	0	5	3	3	1024	32
12.288	0	0	3	3	3	1536	48
12.288	0	0	23	7	3	256	4
12.288	0	0	11	7	3	512	8
12.288	0	0	5	7	3	1024	16
12.288	0	0	3	7	3	1536	24

Table 27-27 shows the example of programming clock controller divider ratio to generate the SSI_MCLK and SSI_BCLK frequencies close to the ideal sampling rates. In these examples, setting the SSI to I^2S master mode (SSI_CR[I2S] = 01) or individually programming the SSI into network, transmit internal clock mode selects the master mode. (The table specifically illustrates the I^2S mode frequencies/sample rates.)

I^2S master mode requires a 32-bit word length, regardless of the actual data type. Consequently, the fixed I^2S frame rate of 64 bits per frame (word length (WL) can be any value) and DC = 1 are assumed.

Table 27-27. SSI Sys Clock, Bit Clock, Frame Clock in Master Mode

Sampling /Frame rate (kHz)	Over-sampling rate	SSI_CLKIN freq (MHz) (SSI_MCLK)	SSI_CCR			Bit Clk (kHz) SSI_BCLK
			DIV2	PSR	PM	
44.10	384	16.934	0	0	2	2822.33
22.05	384	16.934	0	0	5	1411.17
11.025	384	16.934	0	0	11	705.58
48.00	256	12.288	0	0	1	3072

27.4.3 External Frame and Clock Operation

When applying external frame sync and clock signals to the SSI module, at least four bit clock cycles should exist between the enabling of the transmit or receive section and the rising edge of the corresponding frame sync signal. The transition of SSI_FS should be synchronized with the rising edge of external clock signal, SSI_BCLK.

27.4.4 Supported Data Alignment Formats

The SSI supports three data formats to provide flexibility with managing data. These formats dictate how data is written to and read from the data registers. Therefore, data can appear in different places in SSI_TX0/1 and SSI_RX0/1 based on the data format and the number of bits per word. Independent data formats are supported for the transmitter and receiver (i.e. the transmitter and receiver can use different data formats).

The supported data formats are:

- msb alignment
- lsb alignment
  - — Zero-extended (receive data only)
  - — Sign-extended (receive data only)

With msb alignment, the most significant byte is bits 31–24 of the data register if the word length is larger than, or equal to, 16 bits. If the word length is less than 16 bits and msb alignment is chosen, the most significant byte is bits 15–8. With lsb alignment, the least significant byte is bits 7–0. The SSI_TCR[TXBIT0] and the SSI_RCR[RXBIT0] bits control data alignment. Table 27-28 shows the bit assignment for all the data formats supported by the SSI module.

Table 27-28. Data Alignment

Format	Bit Number																														
	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
8-bit lsb Aligned																									7	6	5	4	3	2	1	0
8-bit msb Aligned																	7	6	5	4	3	2	1	0								
10-bit lsb Aligned																							9	8	7	6	5	4	3	2	1	0
10-bit msb Aligned																	9	8	7	6	5	4	3	2	1	0						
12-bit lsb Aligned																					11	10	9	8	7	6	5	4	3	2	1	0
12-bit msb Aligned																	11	10	9	8	7	6	5	4	3	2	1	0				
16-bit lsb Aligned																	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-bit msb Aligned	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																
18-bit lsb Aligned															17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18-bit msb Aligned	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
20-bit lsb Aligned													19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
20-bit msb Aligned	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0												
22-bit lsb Aligned											21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
22-bit msb Aligned	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0										
24-bit lsb Aligned									23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
24-bit msb Aligned	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								

In addition, if lsb alignment is selected, the receive data can be zero-extended or sign-extended.

- In zero-extension, all bits above the most significant bit are 0s. This format is useful when data is stored in a pure integer format.
- In sign-extension, all bits above the most significant bit are equal to the most significant bit. This format is useful when data is stored in a fixed-point integer format (which implies fractional values).

The SSI_RCR[RXEXT] bit controls receive data extension. Transmit data used with lsb alignment has no concept of sign/zero-extension. Unused bits above the most significant bit are simply ignored.

When configured in I^2S or AC97 mode, the SSI forces the lsb alignment. However, the SSI_RCR[RXEXT] bit chooses zero-extension or sign-extension.

Refer to Section 27.3.10, "SSI Transmit Configuration Register (SSI_TCR)," and Section 27.3.11, "SSI Receive Configuration Register (SSI_RCR)," for more detail on the relevant bits in the SSI_TCR and SSI_RCR registers.

27.4.5 Receive Interrupt Enable Bit Description

If the receive FIFO is not enabled, an interrupt occurs when the corresponding SSI receive data ready (SSI_ISR[RDR0/1]) bit is set. If the receive FIFO is enabled and the RIE and RE bit are set, the processor is interrupted when either of the SSI receives FIFO full (SSI_ISR[RFF0/1]) bits is set. When the receive FIFO is enabled, a maximum of eight values are available to be read (eight values per channel in two-channel mode). If not enabled, one value can be read from the SSI_RX register (one each in two-channel mode).

If the RIE bit is cleared, these interrupts are disabled. However, the RFF0/1 and RDR0/1 bits indicate the receive data register full condition. Reading the SSI_RX registers clears the RDR bits, thus clearing the pending interrupt. Two receive data interrupts (two per channel in two-channel mode) are available: receive data with exception status and receive data without exception. Table 27-29 shows the conditions these interrupts are generated.

Table 27-29. SSI Receive Data Interrupts

Interrupt	RIE	ROE*n*	RFF*n*/RDR*n*
Receive Data 0 Interrupts (*n* = 0)			
Receive Data 0 (with exception status)	1	1	1
Receive Data 0 (without exception)	1	0	1
Receive Data 1 Interrupts (*n* = 1)			
Receive Data 1 (with exception status)	1	1	1
Receive Data 1 (without exception)	1	0	1

27.4.6 Transmit Interrupt Enable Bit Description

The SSI transmit interrupt enable (TIE) bit controls interrupts for the SSI transmitter. If the transmit FIFO is enabled and the TIE and TE bits are set, the processor is interrupted when either of the SSI transmit FIFO empty (SSI_ISR[TFE0/1]) flags is set. If the corresponding transmit FIFO is not enabled, an interrupt is generated when the corresponding SSI_ISR[TDE0/1] flag is set and transmit enable (TE) bit is set.

When transmit FIFO 0 is enabled, a maximum of eight values can be written to the SSI (eight per channel in two-channel mode using Tx FIFO 1). If not enabled, then one value can be written to the SSI_TX0 register (one per channel in two-channel mode using SSI_TX1). When the TIE bit is cleared, all transmit interrupts are disabled. However, the TDE0/1 bits always indicate the corresponding SSI_TX register empty condition, even when the transmitter is disabled by the transmit enable (SSI_CR[TE]) bit. Writing data to the SSI_TX clears the corresponding TDE bit, thus clearing the interrupt.

Two transmit data interrupts are available (two per channel in two-Channel mode): transmit data with exception status and transmit data without exceptions. Table 27-30 shows the conditions under which these interrupts are generated.

Table 27-30. SSI Transmit Data Interrupts

Interrupt	TIE	TUE*n*	TFE*n*/TDE*n*
Transmit Data 0 Interrupts (*n* = 0)			
Transmit Data 1 (with exception status)	1	1	1
Transmit Data 1 (without exception)	1	0	1
Transmit Data 1 Interrupts (*n* = 1)			
Transmit Data 0 (with exception status)	1	1	1
Transmit Data 0 (without exception)	1	0	1

27.5 Initialization/Application Information

The following types of reset affected the SSI:

- Power-on reset—Asserting the $\overline{\text{RESET}}$ signal generates the power-on reset. This reset clears the SSI_CR[SSI_EN] bit, which disables the SSI. All other status and control bits in the SSI are affected as described in Table 27-4
- SSI reset—The SSI reset is generated when the SSI_CR[SSI_EN] bit is cleared. The SSI status bits are reset to the same state produced by the power-on reset. The SSI control bits, including those in SSI_CR, are unaffected. The SSI reset is useful for selective reset of the SSI, without changing the present SSI control bits and without affecting the other peripherals.

The correct sequence to initialize the SSI is:

1. Issue a power-on or SSI reset (SSI_CR[SSI_EN] = 0).
2. Set all control bits for configuring the SSI (refer to Table 27-31).
3. Enable appropriate interrupts/DMA requests through SSI_IER.
4. Set the SSI_CR[SSI_EN] bit to enable the SSI.
5. For AC97 mode, set the SSI_ACR[AC97EN] bit after programming the SSI_ATAG register (if needed, for AC97 fixed mode).
6. Set SSI_CR[TE/RE] bits.

To ensure proper operation of the SSI, use the power-on or SSI reset before changing any of the control bits listed in Table 27-31.

NOTE

These control bits should not be changed when the SSI module is enabled.

Table 27-31. SSI Control Bits Requiring SSI to be Disabled Before Change

Control Register	Bit
SSI_CR	[9]=CIS [8]=TCH [7]=MCE [6:5]=I2S [4]=SYN [3]=NET
SSI_IER	[22]=RDMAE [20]=TDMAE
SSI_RCR SSI_TCR	[9]=RXBIT0 and TXBIT0 [8]=RFEN1 and TFEN1 [7]=RFEN0 and TFEN0 [6]=TFDIR [5]=RXDIR and TXDIR [4]=RSHFD and TSHFD [3]=RSCKP and TSCKP [2]=RFSI and TFSI [1]=RFSL and TFSL [0]=REFS and TEFS
SSI_CCR	[16:13]=WL
SSI_ACR	[1]=FV [10:5]=FRDIV

Chapter 28
Real-Time Clock

28.1 Introduction

Figure 28-1 is a block diagram of the functional organization of the real time clock (RTC) module, consisting of:

- Clock Generation
- Time-of-day (TOD) clock counter
- Alarm
- Sampling timer
- Minute stopwatch
- Associated control and bus interface hardware

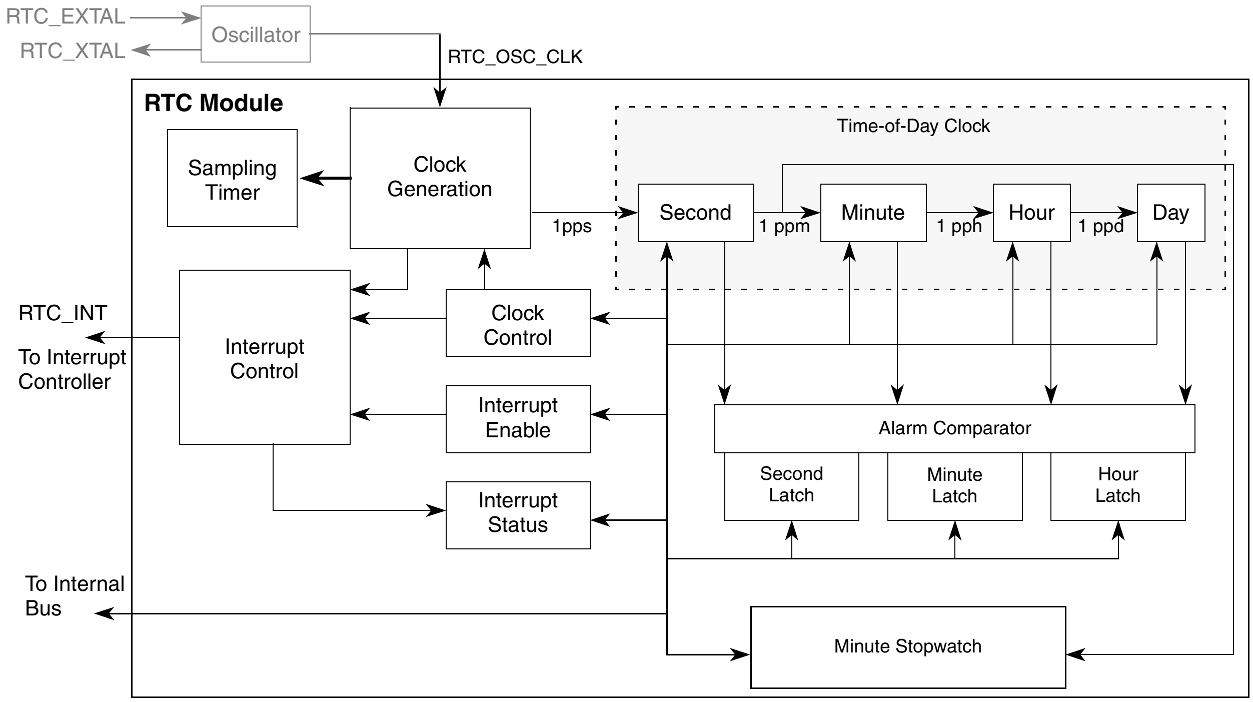


Figure 28-1. Real Time Clock Block Diagram

28.1.1 Overview

This section discusses how to operate and program the real-time clock (RTC) module that maintains a time-of-day clock, provides stopwatch, alarm, and interrupt functions, and supports the following features.

28.1.2 Features

The RTC module includes:

- Full clock—days, hours, minutes, seconds
- Minute countdown timer with interrupt
- Programmable daily alarm with interrupt
- Sampling timer with interrupt
- Once-per-day, once-per-hour, once-per-minute, and once-per-second interrupts
- Operation determined by reference input oscillator clock frequency and value programmed into user-accessible registers
  - Minimum supported oscillator frequency is 2 Hz
- Ability to wake the processor from low-power modes (wait, doze, and stop) via the RTC interrupts

NOTE

The RTC is enabled during stop mode.

28.1.3 Modes of Operation

The real-time-clock operates in various modes as described below:

- Time-of-day counters
  - The clock generation logic divides the reference clock down to 1 Hz using the dividers in the RTC_GOCU and RTC_GOCL registers.
  - The 1 Hz clock increments four counters that are located in three registers
    - RTC_SECONDS contains the 6-bit seconds counter
    - RTC_HOURMIN contains the 6-bit minutes counter and 5-bit hours counter
    - RTC_DAYS contains the 16-bit day counter
- Alarm
  - There are three alarm registers that mirror the three counter registers. An alarm is set by accessing the real-time clock alarm registers (RTC_ALRM_SEC, RTC_ALRM_HM, and RTC_ALRM_DAY) and loading the time minus one second that the alarm should generate an interrupt. When the TOD clock value and the alarm value coincide, one second later an interrupt occurs.
- Sampling Timer
  - The clock generation logic divides the reference clock down to 512 Hz using the dividers in the RTC_GOCU and RTC_GOCL registers. The sampling timer generates a periodic interrupt with frequencies specified by the RTC_IER[SAM*n*,2HZ] bits. This timer can be used for digitizer sampling, keyboard debouncing, or communication polling. Table 28-15 lists some examples of the interrupt frequencies of the sampling timer for the possible reference clocks. Sampling frequencies are dependent upon the RTC oscillator frequency and the value in RTC_GOC[31:9].
- Minute Stopwatch

— The minute stopwatch performs a countdown with a one minute resolution. It generates an interrupt on a minute boundary.

28.2 External Signal Description

The below table describes the RTC external signals.

Table 28-1. RTC Signals

Signal Name	Abbreviation	Function	I/O
RTC External Clock In	RTC_EXTAL	Crystal input clock.	I
RTC Crystal	RTC_XTAL	Oscillator output to crystal.	O

28.3 Memory Map/Register Definition

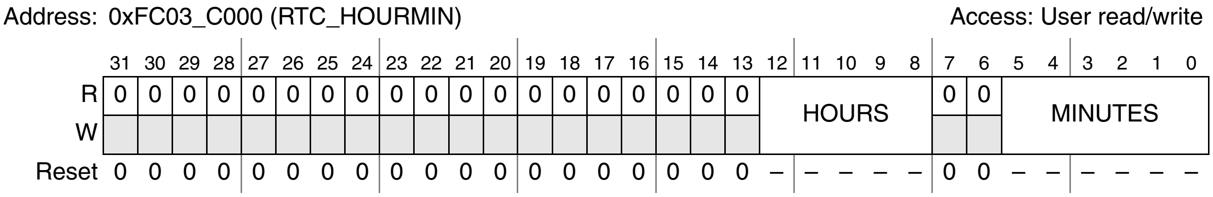
The RTC module includes twelve registers, which are summarized below.

Table 28-2. Real Time Clock Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_C000	RTC Hours and Minutes Counter Register (RTC_HOURMIN)	32	R/W	Undefined	28.3.1/28-3
0xFC03_C004	RTC Seconds Counter Register (RTC_SECONDS)	32	R/W	Undefined	28.3.2/28-4
0xFC03_C008	RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)	32	R/W	0x0000_0000	28.3.3/28-4
0xFC03_C00C	RTC Seconds Alarm Register (RTC_ALRM_SEC)	32	R/W	0x0000_0000	28.3.4/28-5
0xFC03_C010	RTC Control Register (RTC_CR)	32	R/W	0x0000_0080	28.3.5/28-6
0xFC03_C014	RTC Interrupt Status Register (RTC_ISR)	32	R/W	0x0000_0000	28.3.6/28-6
0xFC03_C018	RTC Interrupt Enable Register (RTC_IER)	32	R/W	0x0000_0000	28.3.7/28-7
0xFC03_C01C	Stopwatch Minutes Register (RTC_STPWCH)	32	R/W	0x0000_003F	28.3.8/28-9
0xFC03_C020	RTC Days Counter Register (RTC_DAYS)	32	R/W	0x0000_0000	28.3.9/28-9
0xFC03_C024	RTC Days Alarm Register (RTC_ALRM_DAY)	32	R/W	0x0000_0000	28.3.10/28-9
0xFC03_C034	RTC General Oscillator Clock Upper (RTC_GOCU)	32	R/W	0x0000_0000	28.3.11/28-10
0xFC03_C038	RTC General Oscillator Clock Lower (RTC_GOCL)	32	R/W	0x0000_0000	28.3.12/28-10

28.3.1 RTC Hours and Minutes Counter Register (RTC_HOURMIN)

This register programs the hours and minutes for the TOD clock. It can be read or written at any time. After a write, the time changes to the new value. This register cannot be reset because the real-time clock is always enabled at reset.



Address: 0xFC03_C000 (RTC_HOURMIN) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HOURS					0	0	MINUTES					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	–	–	–	–	–	0	0	–	–	–	–	–	–

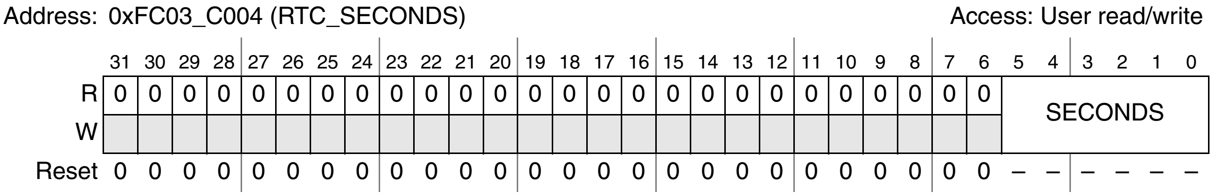
Figure 28-2. RTC Hours and Minutes Counter Register (RTC_HOURMIN)

Table 28-3. RTC_HOURMIN Field Descriptions

Field	Description
31–13	Reserved, must be cleared.
12–8 HOURS	Current hour. Set to any value between 0 and 23 (0x17).
7–6	Reserved, must be cleared.
5–0 MINUTES	Current minutes. Set to any value between 0 and 59 (0x3B).

28.3.2 RTC Seconds Counter Register (RTC_SECONDS)

The real-time clock seconds register (RTC_SECONDS) programs the seconds for the TOD clock. It can be read or written at any time. After a write, the time changes to the new value. This register cannot be reset because real-time clock is always enabled at reset.



Address: 0xFC03_C004 (RTC_SECONDS) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SECONDS					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	–	–	–	–	–	–

Figure 28-3. RTC Seconds Counter Register (RTC_SECONDS)

Table 28-4. RTC_SECONDS Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5–0 SECONDS	Current seconds. Set to any value between 0 and 59 (0x3B).

28.3.3 RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)

The RTC_ALRM_HM register configures the hours and minutes setting for the alarm. The alarm settings can be read or written at any time.

Address: 0xFC03_C008 (RTC_ALRM_HM) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HOURS					0	0	MINUTES					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-4. RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)

Table 28-5. RTC_ALRM_HM Field Descriptions

Field	Description
31–13	Reserved, must be cleared.
12–8 HOURS	Hours setting of the alarm. Set to any value between 0 and 23 (0x17).
7–6	Reserved, must be cleared.
5–0 MINUTES	Minutes setting of the alarm. Set to any value between 0 and 59 (0x3B).

28.3.4 RTC Seconds Alarm Register (RTC_ALRM_SEC)

The RTC_ALRM_SEC register configures the seconds setting for the alarm. The value written to this field must be the alarm time desired minus one second. If the desired alarm time is RTC_ALRM_HM does not equal 0 and RTC_ALRM_SEC equals 0, this affects the RTC_ALRM_HM register as well. The alarm settings can be read or written at any time.

Address: 0xFC03_C00C (RTC_ALRM_SEC) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SECONDS					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. RTC Seconds Alarm Register (RTC_ALRM_SEC)

Table 28-6. RTC_ALRM_SEC Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5–0 SECONDS	Seconds setting of the alarm. The value written to this field must be the alarm time desired minus one second. Set to any value between 0 and 59 (0x3B).

28.3.5 RTC Control Register (RTC_CR)

The RTC_CR register enables the real-time clock module and software reset.

Address: 0xFC03_C010 (RTC_CR) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN	0	0	0	0	0	0	SWR
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 28-6. RTC Control Register (RTC_CR)

Table 28-7. RTC_CR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 EN	RTC enable. Enables/disables the real-time clock module. SWR has no effect on this bit. 0 Disable the RTC 1 Enable the RTC
6–1	Reserved, must be cleared.
0 SWR	Software reset. Resets the module to its default state. The EN bit is also reset to its default value of one. 0 No effect 1 Reset the module

28.3.6 RTC Interrupt Status Register (RTC_ISR)

The real-time clock interrupt status register (RTC_ISR) indicates the status of the various real-time clock interrupts. When an event of the types included in this register occurs, then the bit is set in this register regardless of its corresponding interrupt enable bit. These bits are cleared by writing a value of 1, which also clears the interrupt. Interrupts may occur while the system clock is idle or in sleep mode.

Address: 0xFC03_C014 (RTC_ISR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SAM7	SAM6	SAM5	SAM4	SAM3	SAM2	SAM1	SAM0	2HZ	0	HR	1HZ	DAY	ALM	MIN	SW
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-7. RTC Interrupt Status Register (RTC_ISR)

Table 28-8. RTC_ISR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 SAM*n*	Sampling timer 7–0 interrupt flags. Indicates an interrupt has occurred at the corresponding sampling rate, equal to or close to 2^{n+2} Hz depending on the combination of the RTC oscillator frequency and the programmed value in RTC_GOC[31:9]. See Section 28.4.3, “Sampling Timer,” for more details. 0 No SAM7–0 interrupt has occurred 1 A SAM7–0 interrupt has occurred
7 2HZ	2 Hz interrupt flag. Indicates an interrupt has occurred. If enabled, this bit is set every half a second. 0 No interrupt has occurred 1 A 2 Hz interrupt has occurred
6	Reserved, must be cleared.
5 HR	Hour interrupt flag. If enabled, this bit is set on every increment of the hour counter in the RTC_HOURMIN register. 0 No interrupt has occurred 1 An hour interrupt has occurred
4 1HZ	1 Hz interrupt flag. If enabled, this bit is set on every increment of the second counter of the RTC_SECONDS register. 0 No interrupt has occurred 1 A 1 Hz interrupt has occurred
3 DAY	Day interrupt flag. If enabled, this bit is set on every increment of the day counter in the RTC_DAYS register. 0 No interrupt has occurred 1 A day interrupt has occurred
2 ALM	Alarm interrupt flag. Indicates the real-time clock matches the value in the alarm registers plus one second. The alarm reoccurs every 65,536 days. For a single alarm, clear the interrupt enable for this bit in the interrupt service routine. 0 No interrupt has occurred 1 An alarm interrupt has occurred
1 MIN	If enabled, this bit is set on every increment of the minute counter in the RTC_HOURMIN register. 0 No interrupt has occurred 1 A minute interrupt has occurred
0 SW	Stopwatch flag. Indicates that the stopwatch countdown timed out. 0 The stopwatch did not timeout 1 The stopwatch timed out

28.3.7 RTC Interrupt Enable Register (RTC_IER)

The RTC_IER register enables/disables the various real-time clock interrupts. Masking an interrupt bit has no effect on its corresponding status bit.

Address: 0xFC03_C018 (RTC_IER)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SAM7	SAM6	SAM5	SAM4	SAM3	SAM2	SAM1	SAM0	2HZ	0	HR	1HZ	DAY	ALM	MIN	SW
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. RTC Interrupt Enable Register (RTC_IER)

Table 28-9. RTC_IER Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 SAM7–0	Sampling timer 7–0 interrupt enable. 0 SAM7–0 interrupt disabled 1 SAM7–0 interrupt enabled
7 2HZ	2 Hz interrupt enable. 0 Interrupt disabled 1 2 Hz interrupt enabled
6	Reserved, must be cleared.
5 HR	Hour interrupt enable. 0 Interrupt disabled 1 Hour interrupt enabled
4 1HZ	1 Hz interrupt enable. 0 Interrupt disabled 1 1 Hz interrupt enabled
3 DAY	Day interrupt enable. 0 Interrupt disabled 1 Day interrupt enabled
2 ALM	Alarm interrupt enable. 0 Interrupt disabled 1 Alarm interrupt enabled
1 MIN	Minute interrupt enable. 0 Interrupt disabled 1 Minute interrupt enabled
0 SW	Stopwatch interrupt enable. 0 Interrupt disable 1 Stopwatch interrupt enabled. The stopwatch counts down and remains at -1 (0x3F) until it is reprogrammed. If this bit is enabled with RTC_STPWCF equal to 0x3F, an interrupt is requested on the next minute tick.

28.3.8 RTC Stopwatch Minutes Register (RTC_STPWCH)

The stopwatch minutes register contains the current stopwatch countdown value. When the minute counter of the TOD clock increments, value in this register decrements.

Address: 0xFC03_C01C (RTC_STPWCH) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CNT					
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 28-9. RTC Stopwatch Minutes Register (RTC_STPWCH)

Table 28-10. RTC_STPWCH Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5–0 CNT	Stopwatch count. Contains the stopwatch countdown value plus one minute. Stopwatch counter decrements by the minute (MIN) tick output from the RTC_HOURMIN register, so the average tolerance of the count is 0.5 minutes. For better accuracy, enable the stopwatch by polling the RTC_ISR[MIN] bit or via the minute interrupt service routine. **Note:** Write the value of one less than the desired stopwatch timeout.

28.3.9 RTC Days Counter Register (RTC_DAYS)

The RTC_DAYS register programs the day for the TOD clock. When the RTC_HOURMIN[HOUR] field rolls over from 23 to 0, the day counter increments. It can be read or written at any time. After a write, the time changes to the new value. This register cannot be reset because the real-time clock is always enabled at reset. Only 16-bit accesses to this register are allowed.

Address: 0xFC03_C020 (RTC_DAYS) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DAYS															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-10. RTC Days Counter Register (RTC_DAYS)

Table 28-11. RTC_DAYS Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 DAYS	Current day count. Set to any value between 0 and 65,535 (0xFFFF).

28.3.10 RTC Day Alarm Register (RTC_ALRM_DAY)

The RTC_ALRM_DAY register configures the day for the alarm. The alarm settings can be read or written at any time.

Address: 0xFC03_C024 (RTC_ALRM_DAY) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DAYS															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-11. RTC Day Alarm Register (RTC_ALRM_DAY)

Table 28-12. RTC_ALM_DAYS Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 DAYS	Current day setting of the alarm. Set to any value between 0 and 65,535 (0xFFFF).

28.3.11 RTC General Oscillator Clock Upper Register (RTC_GOCU)

The RTC_GOCU register is the upper two bytes of the 32-bit count value (RTC_GOC) used with the input RTC oscillator clock to create the 1 Hz and sample frequencies. This register can be read or written at any time. A non-zero value must be programmed into RTC_GOC for the 1 Hz internal clock to function.

Address: 0xFC03_C034 (RTC_GOCU) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CNT_UPPER															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-12. RTC General Oscillator Clock Upper Register (RTC_GOCU)

Table 28-13. RTC_GOCU Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 CNT_UPPER	Upper word of RTC_GOC[31:0]; i.e. equal to RTC_GOC[31:16]. RTC_GOC, with the oscillator clock, determines the 1 Hz and sample frequencies. A value of 0 in RTC_GOC turns off the 1 Hz signal to the counters. This field resets to 0.

28.3.12 RTC General Oscillator Clock Lower Register (RTC_GOCL)

The RTC_GOCL register is used as the lower two bytes of the 32-bit count value (RTC_GOC) used with the input RTC oscillator clock to create a 1 Hz and sample frequencies. This register can be read or written at any time. A non-zero value must be programmed into RTC_GOC for the 1 Hz internal clock to function.

Address: 0xFC03_C038 (RTC_GOCL) Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CNT_LOWER															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-13. RTC General Oscillator Clock Lower Register (RTC_GOCL)

Table 28-14. RTC_GOCU Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 CNT_LOWER	Lower word of RTC_GOC[31:0]; i.e. equal to RTC_GOC[15:0]. RTC_GOC, with the oscillator clock, determines the 1 Hz and sample frequencies. A value of 0 in RTC_GOC yields no 1 Hz signal. This register resets to 0.

28.4 Functional Description

The clock generation logic, using the incoming RTC oscillator clock and RTC_GOC value, creates a 1 Hz signal which increments the seconds, minutes, hours, and days counters. The alarm functions, when enabled, generate RTC interrupts when the time-of-day (TOD) settings reach programmed values. The sampling timer generates fixed-frequency interrupts, and the minute stopwatch allows for efficient interrupts on minute boundaries.

28.4.1 Clock Generation and Counter

The clock generation logic divides the reference clock by the value programmed into the RTC_GOC register to obtain a 1 Hz signal. The RTC_GOC is a concatenation of the lower two bytes of the RTC_GOCU and RTC_GOCL registers (RTC_GOCU[15:0] || RTC_GOCL[15:0]). For example:

> With an RTC input clock of 32 kHz, set RTC_GOC to 0x0000_7D00 (32,000).
> RTC_GOCU = 0x0000 and RTC_GOCL = 0x7D00.
>
> With an RTC input clock of 48 kHz, set RTC_GOC to 0x0000_BB80 (48,000).
> RTC_GOCU = 0x0000 and RTC_GOCL = 0xBB80.

The counter portion of the RTC module consists of four groups of counters that are physically located in three registers:

- The 6-bit seconds counter is located in RTC_SECONDS
- The 6-bit minutes counter and the 5-bit hours counter are located in RTC_HOURMIN
- The 16-bit day counter is located in RTC_DAYS

These counters cover a 24-hour clock over 65,536 days. All three registers can be read or written at any time.

Interrupts signal when each of the four counters increments and can indicate when a counter rolls over. For example, each tick of the seconds counter causes the 1HZ interrupt flag to set. When the seconds counter rolls from 59 to 00, the minute counter increments and the MIN interrupt flag is set. The same is true for the minute counter with the HR signal and the hour counter with the DAY signal.

28.4.2 Alarm

There are three alarm registers that mirror the three counter registers. An alarm is set by accessing the real-time clock alarm registers (RTC_ALRM_HM, RTC_ALRM_SEC, and RTC_ALRM_DAY) and loading the time minus one second that the alarm must generate an interrupt. If the RTC_IER[ALM] bit is set when the TOD clock value and the alarm value coincide an interrupt occurs one second later. If the alarm is not disabled and programmed, an alarm reoccurs every 65,536 days. If a single alarm is desired, the alarm function must be disabled through the RTC_IER register during the alarm interrupt service routine.

See Section 28.5, "Initialization/Application Information," for the correct procedure to follow when changing the alarm or time-of-day (day, hour, minute, or second) registers.

28.4.3 Sampling Timer

The sampling timer supports application software. The sampling timer generates a periodic interrupt with the frequency specified by RTC_IER[SAM*n*,2HZ]. This timer can be used for digitizer sampling, keyboard debouncing, or communication polling. The sampling timer operates only if the real-time clock is enabled and the 1 Hz signal is programmed to clock at 1 Hz. The sample clock (which is equal to SAM7) is generated by dividing the RTC oscillator frequency by the value programmed into RTC_GOC[31:9], which is equal to {RTC_GOCU[15:0], RTC_GOCL[15:9]}.

The following table lists example interrupt frequencies of the sampling timer for possible combinations of RTC oscillator frequency and RTC_GOC values. The following definitions apply:

RTC_OSC = RTC oscillator frequency

RTC_GOC = RTC_GOCU[15:0] + RTC_GOCL[15:0]

SAMPLE_COUNT = RTC_GOCU[15:0] + RTC_GOCL[15:9]

Multiple RTC_IER[SAM*n*,2HZ] bits may be set and the corresponding bits in the RTC_ISR register are set at the noted frequencies.

Table 28-15. Example Sampling Timer Frequencies

Sampling Frequency	RTC_OSC = 32 kHz
	RTC_GOC = 0x7D00
	SAMPLE_COUNT = 0x3E
		RTC_OSC = 32.768 kHz
		RTC_GOC = 0x8000
		SAMPLE_COUNT = 0x40
			RTC_OSC = 38.4 kHz
			RTC_GOC = 0x9600
			SAMPLE_COUNT = 0x4B
				RTC_OSC = 48 kHz
				RTC_GOC = 0xBB80
				SAMPLE_COUNT = 0x5D

SAM7	516.13 Hz	512.00 Hz	512.00 Hz	516.13 Hz
SAM6	258.06 Hz	256.00 Hz	256.00 Hz	258.06 Hz
SAM5	129.03 Hz	128.00 Hz	128.00 Hz	129.03 Hz
SAM4	64.52 Hz	64.00 Hz	64.00 Hz	64.52 Hz
SAM3	32.26Hz	32.00 Hz	32.00 Hz	32.26Hz
SAM2	16.13 Hz	16.00 Hz	16.00 Hz	16.13
SAM1	8.06 Hz	8.00 Hz	8.00 Hz	8.06 Hz
SAM0	4.03 Hz	4.00 Hz	4.00 Hz	4.03 Hz
2HZ	2.02 Hz	2.00 Hz	2.00 Hz	2.02 Hz

28.4.4 Minute Stopwatch

The minute stopwatch performs a countdown with a one minute resolution. It can generate an interrupt on a minute boundary. For example, to turn off a peripheral after five minutes of inactivity, program a value of 0x04 into RTC_STPWCH[CNT]. At each minute, the value in the stopwatch decrements. When the stopwatch value reaches -1, interrupt occurs. The value of the register does not change until it is reprogrammed. The actual delay includes the seconds from setting the stopwatch to the next minute tick.

28.5 Initialization/Application Information

28.5.1 Flow Chart of RTC Operation

Table 28-14 shows the flow chart of a typical RTC operation.

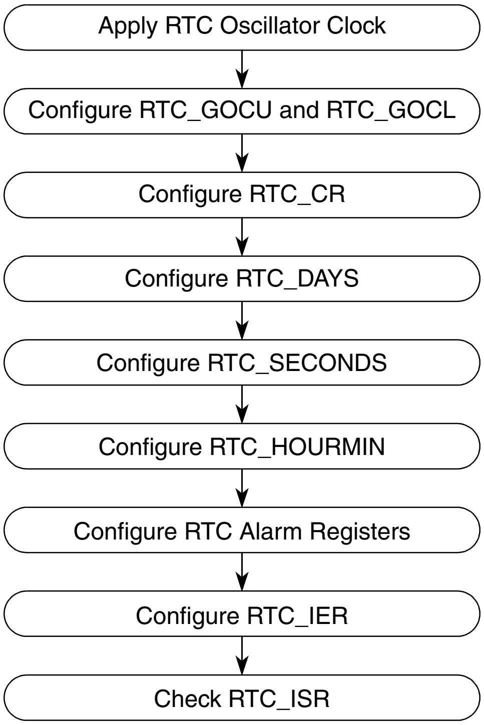


Figure 28-14. Flow Chart of RTC Operation

28.5.2 Programming the Alarm or Time-of-Day Registers

Use the following procedure illustrated in Figure 28-15 when changing the alarm or time-of-day (day, hour, minute, and second) registers.

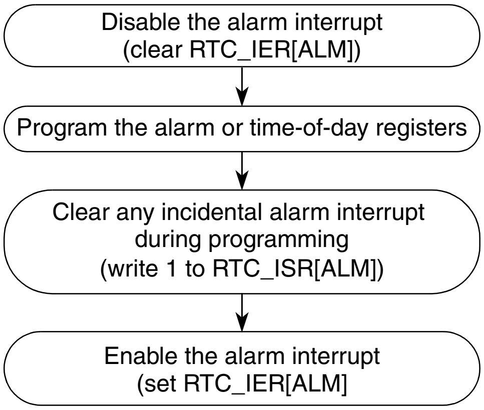

Disable the alarm interrupt
(clear RTC_IER[ALM])
Program the alarm or time-of-day registers
Clear any incidental alarm interrupt
during programming
(write 1 to RTC_ISR[ALM])
Enable the alarm interrupt
(set RTC_IER[ALM]

Figure 28-15. Flow Chart of Alarm and Time-of-Day Programming

Chapter 29
Programmable Interrupt Timers (PIT0–PIT3)

29.1 Introduction

This chapter describes the operation of the four programmable interrupt timer modules: PIT0–PIT3.

29.1.1 Overview

Each PIT is a 16-bit timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can count down from the value written in the modulus register or it can be a free-running down-counter.

29.1.2 Block Diagram

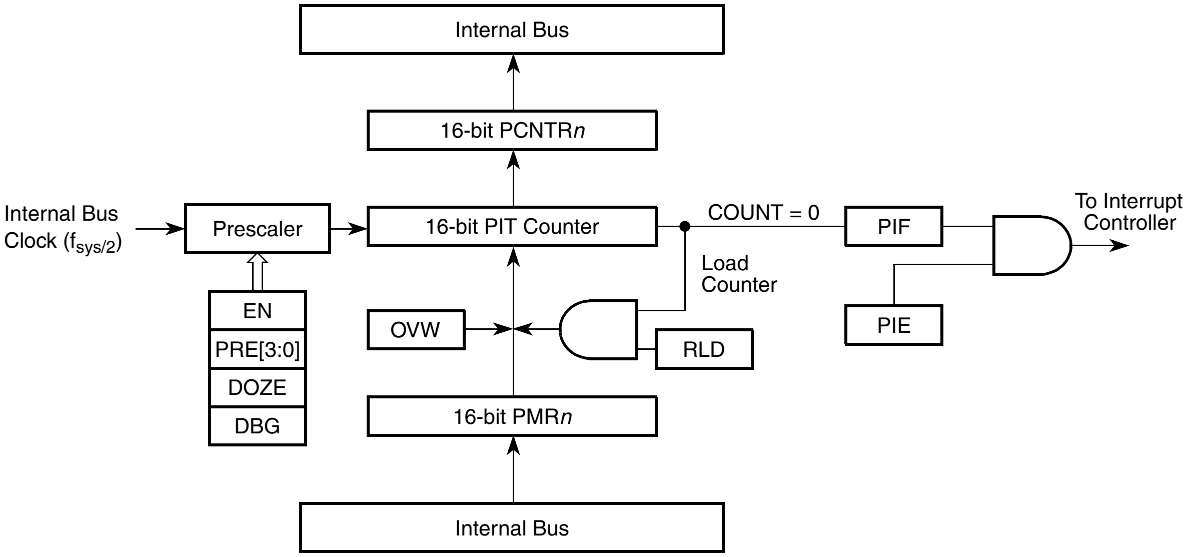


Figure 29-1. PIT Block Diagram

29.1.3 Low-Power Mode Operation

This subsection describes the operation of the PIT modules in low-power modes and debug mode of operation. Low-power modes are described in the power management module, Chapter 9, “Power Management.” Table 29-1 shows the PIT module operation in low-power modes and how it can exit from each mode.

NOTE

The low-power interrupt control register (LPICR) in the system control module specifies the interrupt level at or above which the device can be brought out of a low-power mode.

Table 29-1. PIT Module Operation in Low-power Modes

Low-power Mode	PIT Operation	Mode Exit
Wait	Normal	N/A
Doze	Normal if PCSR*n*[DOZE] cleared, stopped otherwise	Any interrupt at or above level in LPICR, exit doze mode if PCSR*n*[DOZE] is set. Otherwise interrupt assertion has no effect.
Stop	Stopped	No
Debug	Normal if PCSR*n*[DBG] cleared, stopped otherwise	No. Any interrupt is serviced upon normal exit from debug mode

In wait mode, the PIT module continues to operate as in run mode and can be configured to exit the low-power mode by generating an interrupt request. In doze mode with the PCSR*n*[DOZE] bit set, PIT module operation stops. In doze mode with the PCSR*n*[DOZE] bit cleared, doze mode does not affect PIT operation. When doze mode is exited, PIT continues operating in the state it was in prior to doze mode. In stop mode, the internal bus clock is absent and PIT module operation stops.

In debug mode with the PCSR*n*[DBG] bit set, PIT module operation stops. In debug mode with the PCSR*n*[DBG] bit cleared, debug mode does not affect PIT operation. When debug mode is exited, the PIT continues to operate in its pre-debug mode state, but any updates made in debug mode remain.

29.2 Memory Map/Register Definition

This section contains a memory map (see Table 29-2) and describes the register structure for PIT0–PIT3.

NOTE

Longword accesses to any of the programmable interrupt timer registers results in a bus error. Only byte and word accesses are allowed.

Table 29-2. Programmable Interrupt Timer Modules Memory Map

Address	Register	Width (bits)	Access[1]	Reset Value	Section/Page
PIT 0 PIT 1 PIT 2 PIT 3					
Supervisor Access Only Registers[2]					
0xFC08_0000 0xFC08_4000 0xFC08_8000 0xFC08_C000	PIT Control and Status Register (PCSR*n*)	16	R/W	0x0000	29.2.1/29-3

Table 29-2. Programmable Interrupt Timer Modules Memory Map (continued)

Address	Register	Width (bits)	Access[1]	Reset Value	Section/Page
PIT 0 PIT 1 PIT 2 PIT 3					
0xFC08_0002 0xFC08_4002 0xFC08_8002 0xFC08_C002	PIT Modulus Register (PMR*n*)	16	R/W	0xFFFF	29.2.2/29-5
User/Supervisor Access Registers					
0xFC08_0004 0xFC08_4004 0xFC08_8004 0xFC08_C004	PIT Count Register (PCNTR*n*)	16	R	0xFFFF	29.2.3/29-5

[1] Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

[2] User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

29.2.1 PIT Control and Status Register (PCSR*n*)

The PCSR*n* registers configure the corresponding timer's operation.

Address: 0xFC08_0000 (PCSR0)
0xFC08_4000 (PCSR1)
0xFC08_8000 (PCSR2)
0xFC08_C000 (PCSR3)

Access: Supervisor
read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	PRE				0	DOZE	DBG	OVW	PIE	PIF	RLD	EN
W	▨	▨	▨	▨					▨					w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-2. PCSR*n* Register

Table 29-3. PCSR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–12</td><td>Reserved, must be cleared.</td></tr>
<tr><td>11–8
PRE</td><td>Prescaler. The read/write prescaler bits select the internal bus clock divisor to generate the PIT clock. To accurately predict the timing of the next count, change the PRE[3:0] bits only when the enable bit (EN) is clear. Changing PRE[3:0] resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Setting the EN bit and writing to PRE[3:0] can be done in this same write cycle. Clearing the EN bit stops the prescaler counter.
<table>
<tr><th>PRE</th><th>Internal Bus Clock Divisor</th><th>Decimal Equivalent</th></tr>
<tr><td>0000</td><td>2^{0}</td><td>1</td></tr>
<tr><td>0001</td><td>2^{1}</td><td>2</td></tr>
<tr><td>0010</td><td>2^{2}</td><td>4</td></tr>
<tr><td>...</td><td>...</td><td>...</td></tr>
<tr><td>1101</td><td>2^{13}</td><td>8192</td></tr>
<tr><td>1110</td><td>2^{14}</td><td>16384</td></tr>
<tr><td>1111</td><td>2^{15}</td><td>32768</td></tr>
</table>
</td></tr>
<tr><td>7</td><td>Reserved, must be cleared.</td></tr>
<tr><td>6
DOZE</td><td>Doze Mode Bit. The read/write DOZE bit controls the function of the PIT in doze mode. Reset clears DOZE.
0 PIT function not affected in doze mode
1 PIT function stopped in doze mode. When doze mode is exited, timer operation continues from the state it was in before entering doze mode.</td></tr>
<tr><td>5
DBG</td><td>Debug mode bit. Controls the function of PIT in halted/debug mode. Reset clears DBG. During debug mode, register read and write accesses function normally. When debug mode is exited, timer operation continues from the state it was in before entering debug mode, but any updates made in debug mode remain.
0 PIT function not affected in debug mode
1 PIT function stopped in debug mode
<b>Note:</b> Changing the DBG bit from 1 to 0 during debug mode starts the PIT timer. Likewise, changing the DBG bit from 0 to 1 during debug mode stops the PIT timer.</td></tr>
<tr><td>4
OVW</td><td>Overwrite. Enables writing to PMR<i>n</i> to immediately overwrite the value in the PIT counter.
0 Value in PMR<i>n</i> replaces value in PIT counter when count reaches 0x0000.
1 Writing PMR<i>n</i> immediately replaces value in PIT counter.</td></tr>
<tr><td>3
PIE</td><td>PIT interrupt enable. This read/write bit enables PIF flag to generate interrupt requests.
0 PIF interrupt requests disabled
1 PIF interrupt requests enabled</td></tr>
<tr><td>2
PIF</td><td>PIT interrupt flag. This read/write bit is set when PIT counter reaches 0x0000. Clear PIF by writing a 1 to it or by writing to PMR. Writing 0 has no effect. Reset clears PIF.
0 PIT count has not reached 0x0000.
1 PIT count has reached 0x0000.</td></tr>
</table>

Table 29-3. PCSR*n* Field Descriptions (continued)

Field	Description
1 RLD	Reload bit. The read/write reload bit enables loading the value of PMR*n* into PIT counter when the count reaches 0x0000. 0 Counter rolls over to 0xFFFF on count of 0x0000 1 Counter reloaded from PMR*n* on count of 0x0000
0 EN	PIT enable bit. Enables PIT operation. When PIT is disabled, counter and prescaler are held in a stopped state. This bit is read anytime, write anytime. 0 PIT disabled 1 PIT enabled

29.2.2 PIT Modulus Register (PMR*n*)

The 16-bit read/write PMR*n* contains the timer modulus value loaded into the PIT counter when the count reaches 0x0000 and the PCSR*n*[RLD] bit is set.

When the PCSR*n*[OVW] bit is set, PMR*n* is transparent, and the value written to PMR*n* is immediately loaded into the PIT counter. The prescaler counter is reset (0xFFFF) anytime a new value is loaded into the PIT counter and also during reset. Reading the PMR*n* returns the value written in the modulus latch. Reset initializes PMR*n* to 0xFFFF.

Address: 0xFC08_0002 (PMR0)
0xFC08_4002 (PMR1)
0xFC08_8002 (PMR2)
0xFC08_C002 (PMR3)

Access: Supervisor
read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PM															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 29-3. PIT Modulus Register (PMR*n)*

Table 29-4. PMR*n* Field Descriptions

Field	Description
15–0 PM	Timer modulus. The value of this register is loaded into the PIT counter when the count reaches zero and the PCSR*n*[RLD] bit is set. However, if PCSR*n*[OVW] is set, the value written to this field is immediately loaded into the counter. Reading this field returns the value written.

29.2.3 PIT Count Register (PCNTR*n*)

The 16-bit, read-only PCNTR*n* contains the counter value. Reading the 16-bit counter with two 8-bit reads is not guaranteed coherent. Writing to PCNTR*n* has no effect, and write cycles are terminated normally.

Address: 0xFC08_0004 (PCNTR0)
0xFC08_4004 (PCNTR1)
0xFC08_8004 (PCNTR2)
0xFC08_C004 (PCNTR3)

Access: User read only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PC															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 29-4. PIT Count Register (PCNTR*n*)

Table 29-5. PCNTR*n* Field Descriptions

Field	Description
15–0 PC	Counter value. Reading this field with two 8-bit reads is not guaranteed coherent. Writing to PCNTR*n* has no effect, and write cycles are terminated normally.

29.3 Functional Description

This section describes the PIT functional operation.

29.3.1 Set-and-Forget Timer Operation

This mode of operation is selected when the RLD bit in the PCSR register is set.

When PIT counter reaches a count of 0x0000, PIF flag is set in PCSR*n*. The value in the modulus register loads into the counter, and the counter begins decrementing toward 0x0000. If the PCSR*n*[PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSR*n*[OVW] bit is set, the counter can be directly initialized by writing to PMR*n* without having to wait for the count to reach 0x0000.

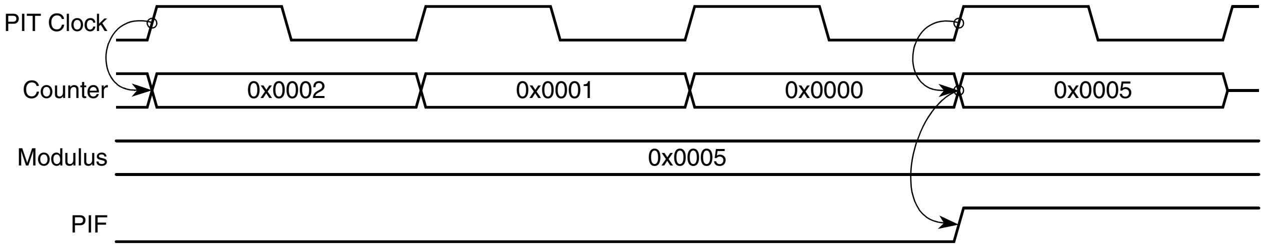


Figure 29-5. Counter Reloading from the Modulus Latch

29.3.2 Free-Running Timer Operation

This mode of operation is selected when the PCSR*n*[RLD] bit is clear. In this mode, the counter rolls over from 0x0000 to 0xFFFF without reloading from the modulus latch and continues to decrement.

When the counter reaches a count of 0x0000, PCSR*n*[PIF] flag is set. If the PCSR*n*[PIE] bit is set, PIF flag issues an interrupt request to the CPU.

When the PCSR*n*[OVW] bit is set, counter can be directly initialized by writing to PMR*n* without having to wait for the count to reach 0x0000.

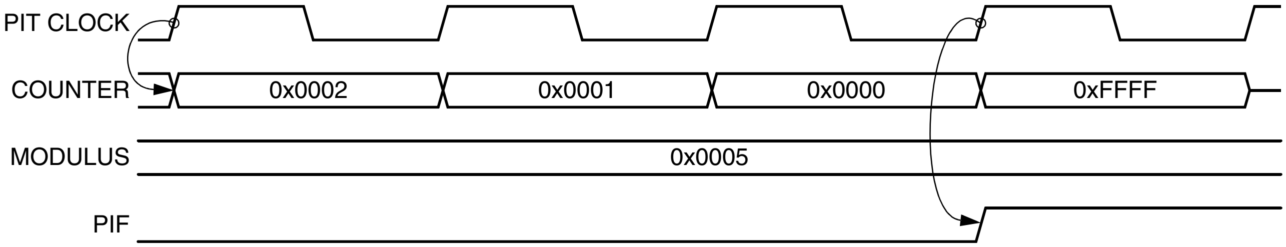


Figure 29-6. Counter in Free-Running Mode

29.3.3 Timeout Specifications

The 16-bit PIT counter and prescaler supports different timeout periods. The prescaler divides the internal bus clock period as selected by the PCSR*n*[PRE] bits. The PMR*n*[PM] bits select the timeout period.

$$\text{Timeout period} = \frac{2^{\text{PCSRn[PRE]}} \times (\text{PMRn[PM]} + 1)}{f_{sys/2}} \qquad \textit{Eqn. 29-1}$$

29.3.4 Interrupt Operation

Table 29-6 shows the interrupt request generated by the PIT.

Table 29-6. PIT Interrupt Requests

Interrupt Request	Flag	Enable Bit
Timeout	PIF	PIE

The PIF flag is set when the PIT counter reaches 0x0000. The PIE bit enables the PIF flag to generate interrupt requests. Clear PIF by writing a 1 to it or by writing to the PMR.

Chapter 30
DMA Timers (DTIM0–DTIM3)

30.1 Introduction

This chapter describes the configuration and operation of the four direct memory access (DMA) timer modules (DTIM0, DTIM1, DTIM2, and DTIM3). These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or DMA triggers. Additionally, programming examples are included.

NOTE

The designation *n* appears throughout this section to refer to registers or signals associated with one of the four identical timer modules: DTIM0, DTIM1, DTIM2, or DTIM3.

30.1.1 Overview

Each DMA timer module has a separate register set for configuration and control. The timers can be configured to operate from the internal bus clock (f_{sys}) or from an external clocking source using the DT*n*IN signal. If the internal bus clock is selected, it can be divided by 16 or 1. The selected clock source is routed to an 8-bit programmable prescaler that clocks the actual DMA timer counter register (DTCN*n*). Using the DTMR*n*, DTXMR*n*, DTCR*n*, and DTRR*n* registers, the DMA timer may be configured to assert an output signal, generate an interrupt, or request a DMA transfer on a particular event.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, “Pin Multiplexing and Control”) prior to configuring the DMA Timers.

Figure 30-1 is a block diagram of one of the four identical timer modules.

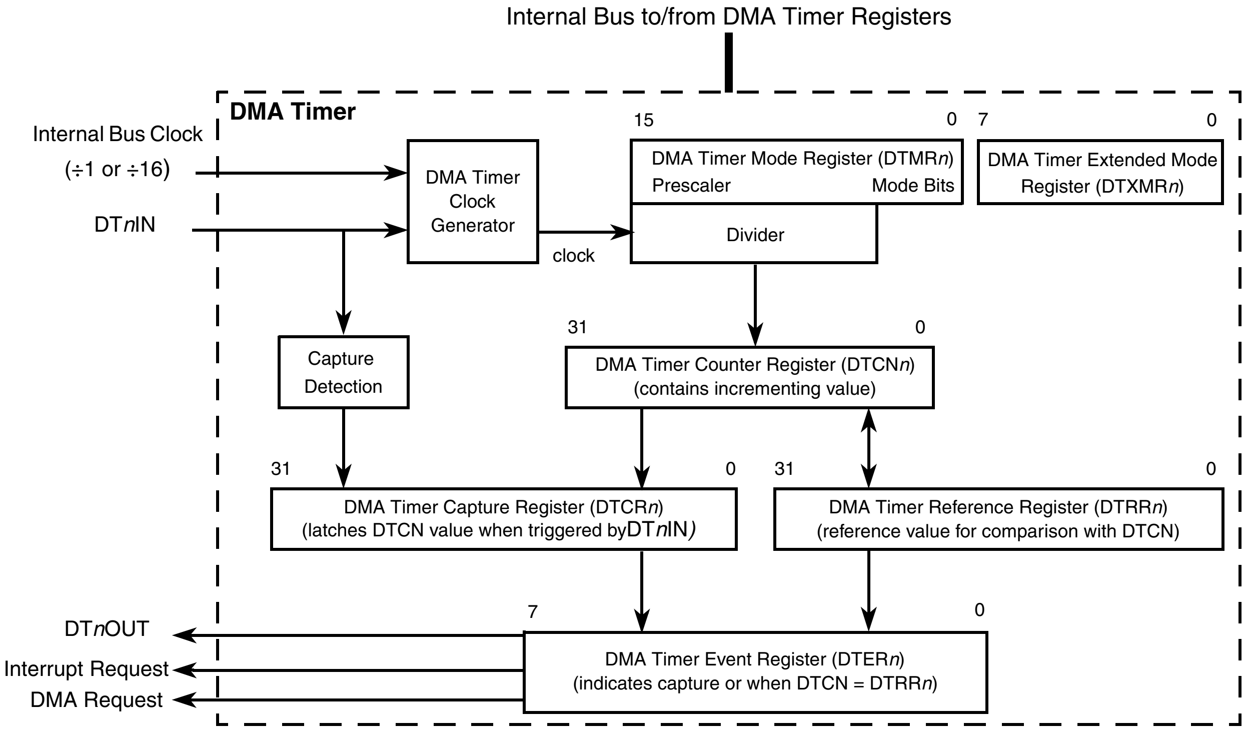

Internal Bus to/from DMA Timer Registers
DMA Timer
Internal Bus Clock
(÷1 or ÷16)
DTnIN
DMA Timer Clock Generator
clock
15
0
DMA Timer Mode Register (DTMRn)
Prescaler
Mode Bits
Divider
7
0
DMA Timer Extended Mode Register (DTXMRn)
Capture Detection
31
0
DMA Timer Counter Register (DTCNn) (contains incrementing value)
31
0
DMA Timer Capture Register (DTCRn) (latches DTCN value when triggered byDTnIN)
31
0
DMA Timer Reference Register (DTRRn) (reference value for comparison with DTCN)
7
0
DMA Timer Event Register (DTERn) (indicates capture or when DTCN = DTRRn)
DTnOUT
Interrupt Request
DMA Request

Figure 30-1. DMA Timer Block Diagram

30.1.2 Features

Each DMA timer module has:

- Maximum timeout period of 132,272 seconds at 133 MHz (~38 hours)
- 7.5-ns resolution at 133 MHz
- Programmable sources for the clock input, including external clock
- Programmable prescaler
- Input-capture capability with programmable trigger edge on input pin
- Programmable mode for the output pin on reference compare
- Free run and restart modes
- Programmable interrupt or DMA request on input capture or reference-compare
- Ability to stop the timer from counting when the ColdFire core is halted

30.2 Memory Map/Register Definition

The timer module registers, shown in Table 30-1, can be modified at any time.

Table 30-1. DMA Timer Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
DMA Timer 0 DMA Timer 1 DMA Timer 2 DMA Timer 3					
0xFC07_0000 0xFC07_4000 0xFC07_8000 0xFC07_C000	DMA Timer *n* Mode Register (DTMR*n*)	16	R/W	0x0000	30.2.1/30-3
0xFC07_0002 0xFC07_4002 0xFC07_8002 0xFC07_C002	DMA Timer *n* Extended Mode Register (DTXMR*n*)	8	R/W	0x00	30.2.2/30-5
0xFC07_0003 0xFC07_4003 0xFC07_8003 0xFC07_C003	DMA Timer *n* Event Register (DTER*n*)	8	R/W	0x00	30.2.3/30-5
0xFC07_0004 0xFC07_4004 0xFC07_8004 0xFC07_C004	DMA Timer *n* Reference Register (DTRR*n*)	32	R/W	0xFFFF_FFFF	30.2.4/30-7
0xFC07_0008 0xFC07_4008 0xFC07_8008 0xFC07_C008	DMA Timer *n* Capture Register (DTCR*n*)	32	R/W	0x0000_0000	30.2.5/30-7
0xFC07_000C 0xFC07_400C 0xFC07_800C 0xFC07_C00C	DMA Timer *n* Counter Register (DTCN*n*)	32	R	0x0000_0000	30.2.6/30-8

30.2.1 DMA Timer Mode Registers (DTMR*n*)

The DTMR*n* registers program the prescaler and various timer modes.

Address: 0xFC07_0000 (DTMR0)
0xFC07_4000 (DTMR1)
0xFC07_8000 (DTMR2)
0xFC07_C000 (DTMR3)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PS								CE		OM	ORRI	FRR	CLK		RST
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-2. DTMR*n* Registers

Table 30-2. DTMR*n* Field Descriptions

Field	Description
15–8 PS	Prescaler value. Divides the clock input (internal bus clock/(16 or 1) or clock on DT*n*IN) 0x00 1 ... 0xFF 256
7–6 CE	Capture edge. 00 Disable capture event output. Timer in reference mode. 01 Capture on rising edge only 10 Capture on falling edge only 11 Capture on any edge
5 OM	Output mode. 0 Active-low pulse for one internal bus clock cycle (7.5-ns resolution at 133 MHz) 1 Toggle output.
4 ORRI	Output reference request, interrupt enable. If ORRI is set when DTER*n*[REF] is set, a DMA request or an interrupt occurs, depending on the value of DTXMR*n*[DMAEN] (DMA request if set, interrupt if cleared). 0 Disable DMA request or interrupt for reference reached (does not affect DMA request or interrupt on capture function). 1 Enable DMA request or interrupt upon reaching the reference value.
3 FRR	Free run/restart 0 Free run. Timer count continues incrementing after reaching the reference value. 1 Restart. Timer count is reset immediately after reaching the reference value.
2–1 CLK	Input clock source for the timer. Avoid setting CLK when RST is already set. Doing so causes CLK to zero (stop counting). 00 Stop count 01 Internal bus clock divided by 1 10 Internal bus clock divided by 16. This clock source is not synchronized with the timer; therefore, successive time-outs may vary ghtly. 11 DT*n*IN pin (falling edge)
0 RST	Reset timer. Performs a software timer reset similar to an external reset, although other register values can be written while RST is cleared. A transition of RST from 1 to 0 resets register values. The timer counter is not clocked unless the timer is enabled. 0 Reset timer (software reset) 1 Enable timer

30.2.2 DMA Timer Extended Mode Registers (DTXMR*n*)

The DTXMR*n* registers program DMA request and increment modes for the timers.

Address: 0xFC07_0002 (DTXMR0)
0xFC07_4002 (DTXMR1)
0xFC07_8002 (DTXMR2)
0xFC07_C002 (DTXMR3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	DMAEN	HALTED	0	0	0	0	0	MODE16
W								
Reset:	0	0	0	0	0	0	0	0

Figure 30-3. DTXMR*n* Registers

Table 30-3. DTXMR*n* Field Descriptions

Field	Description
7 DMAEN	DMA request. Enables DMA request output on counter reference match or capture edge event. 0 DMA request disabled 1 DMA request enabled
6 HALTED	Controls the counter when the core is halted. This allows debug mode to be entered without timer interrupts affecting the debug flow. 0 Timer function is not affected by core halt. 1 Timer stops counting while the core is halted. **Note:** This bit is only applicable in reference compare mode, see Section 30.3.3, “Reference Compare.”
5–1	Reserved, must be cleared.
0 MODE16	Selects the increment mode for the timer. Setting MODE16 is intended to exercise the upper bits of the 32-bit timer in diagnostic software without requiring the timer to count through its entire dynamic range. When set, the counter’s upper 16 bits mirror its lower 16 bits. All 32 bits of the counter remain compared to the reference value. 0 Increment timer by 1 1 Increment timer by 65,537

30.2.3 DMA Timer Event Registers (DTER*n*)

DTER*n*, shown in Figure 30-4, reports capture or reference events by setting DTER*n*[CAP] or DTER*n*[REF]. This reporting happens regardless of the corresponding DMA request or interrupt enable values, DTXMR*n*[DMAEN] and DTMR*n*[ORRI,CE].

Writing a 1 to DTER*n*[REF] or DTER*n*[CAP] clears it (writing a 0 does not affect bit value); both bits can be cleared at the same time. If configured to generate an interrupt request, clear REF and CAP early in the interrupt service routine so the timer module can negate the interrupt request signal to the interrupt controller. If configured to generate a DMA request, processing of the DMA data transfer automatically clears the REF and CAP flags via the internal DMA ACK signal.

Address: 0xFC07_0003 (DTER0)
0xFC07_4003 (DTER1)
0xFC07_8003 (DTER2)
0xFC07_C003 (DTER3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	REF	CAP
W							w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 30-4. DTER*n* Registers

Table 30-4. DTER*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–2</td><td>Reserved, must be cleared.</td></tr>
<tr><td>1
REF</td><td>Output reference event. The counter value (DTCN<i>n</i>) equals DTRR<i>n</i>. Writing a 1 to REF clears the event condition. Writing a 0 has no effect.
<table>
<tr><th>REF</th><th>DTMR<i>n</i>[ORRI]</th><th>DTXMR<i>n</i>[DMAEN]</th><th></th></tr>
<tr><td>0</td><td>X</td><td>X</td><td>No event</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>No request asserted</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>No request asserted</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Interrupt request asserted</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>DMA request asserted</td></tr>
</table></td></tr>
<tr><td>0
CAP</td><td>Capture event. The counter value has been latched into DTCR<i>n</i>. Writing a 1 to CAP clears the event condition. Writing a 0 has no effect.
<table>
<tr><th>CAP</th><th>DTMR<i>n</i>[CE]</th><th>DTXMR<i>n</i>
[DMAEN]</th><th></th></tr>
<tr><td>0</td><td>XX</td><td>X</td><td>No event</td></tr>
<tr><td>1</td><td>00</td><td>0</td><td>Disable capture event output</td></tr>
<tr><td>1</td><td>00</td><td>1</td><td>Disable capture event output</td></tr>
<tr><td>1</td><td>01</td><td>0</td><td>Capture on rising edge and trigger interrupt</td></tr>
<tr><td>1</td><td>01</td><td>1</td><td>Capture on rising edge and trigger DMA</td></tr>
<tr><td>1</td><td>10</td><td>0</td><td>Capture on falling edge and trigger interrupt</td></tr>
<tr><td>1</td><td>10</td><td>1</td><td>Capture on falling edge and trigger DMA</td></tr>
<tr><td>1</td><td>11</td><td>0</td><td>Capture on any edge and trigger interrupt</td></tr>
<tr><td>1</td><td>11</td><td>1</td><td>Capture on any edge and trigger DMA</td></tr>
</table></td></tr>
</table>

30.2.4 DMA Timer Reference Registers (DTRR*n*)

As part of the output-compare function, each DTRR*n* contains the reference value compared with the respective free-running timer counter (DTCN*n*).

The reference value is matched when DTCN*n* equals DTRR*n*. The prescaler indicates that DTCN*n* should be incremented again. Therefore, the reference register is matched after DTRR*n* + 1 time intervals.

Address: 0xFC07_0004 (DTRR0)
0xFC07_4004 (DTRR1)
0xFC07_8004 (DTRR2)
0xFC07_C004 (DTRR3)

Access: User 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
R	REF (32-bit reference value)																															
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 30-5. DTRR*n* Registers

Table 30-5. DTRR*n* Field Descriptions

Field	Description
31–0 REF	Reference value compared with the respective free-running timer counter (DTCN*n*) as part of the output-compare function.

30.2.5 DMA Timer Capture Registers (DTCR*n*)

Each DTCR*n* latches the corresponding DTCN*n* value during a capture operation when an edge occurs on DT*n*IN, as programmed in DTMR*n*. The internal bus clock is assumed to be the clock source. DT*n*IN cannot simultaneously function as a clocking source and as an input capture pin. Indeterminate operation results if DT*n*IN is set as the clock source when the input capture mode is used.

Address: 0xFC07_0008 (DTCR0)
0xFC07_4008 (DTCR1)
0xFC07_8008 (DTCR2)
0xFC07_C008 (DTCR3)

Access: User 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
R	CAP (32-bit capture counter value)																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-6. DTCR*n* Registers

Table 30-6. DTCR*n* Field Descriptions

Field	Description
31–0 CAP	Captures the corresponding DTCN*n* value during a capture operation when an edge occurs on DT*n*IN, as programmed in DTMR*n*.

30.2.6 DMA Timer Counters (DTCN*n*)

The current value of the 32-bit timer counter can be read at anytime without affecting counting. Writes to DTCN*n* clear the timer counter. The timer counter increments on the clock source rising edge (internal bus clock divided by 1, internal bus clock divided by 16, or DT*n*IN).

Address: 0xFC07_000C (DTCN0)
0xFC07_400C (DTCN1)
0xFC07_800C (DTCN2)
0xFC07_C00C (DTCN3)

Access: User 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
R	CNT (32-bit timer counter value count)																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-7. DMA Timer Counters (DTCN*n*)

Table 30-7. DTCN*n* Field Descriptions

Field	Description
31–0 CNT	Timer counter. Can be read at anytime without affecting counting and any write to this field clears it.

30.3 Functional Description

30.3.1 Prescaler

The prescaler clock input is selected from the internal bus clock ($f_{sys/2}$ divided by 1 or 16) or from the corresponding timer input, DT*n*IN. DT*n*IN is synchronized to the internal bus clock, and the synchronization delay is between two and three internal bus clocks. The corresponding DTMR*n*[CLK] selects the clock input source. A programmable prescaler divides the clock input by values from 1 to 256. The prescaler output is an input to the 32-bit counter, DTCN*n*.

30.3.2 Capture Mode

Each DMA timer has a 32-bit timer capture register (DTCR*n*) that latches the counter value when the corresponding input capture edge detector senses a defined DT*n*IN transition. The capture edge bits (DTMR*n*[CE]) select the type of transition that triggers the capture and sets the timer event register capture event bit, DTER*n*[CAP]. If DTER*n*[CAP] and DTXMR*n*[DMAEN] are set, a DMA request is asserted. If DTER*n*[CAP] is set and DTXMR*n*[DMAEN] is cleared, an interrupt is asserted.

30.3.3 Reference Compare

Each DMA timer can be configured to count up to a reference value. If the reference value is met, DTER*n*[REF] is set.

- If DTMR*n*[ORRI] is set and DTXMR*n*[DMAEN] is cleared, an interrupt is asserted.
- If DTMR*n*[ORRI] and DTXMR*n*[DMAEN] are set, a DMA request is asserted.

If the free run/restart bit (DTMR*n*[FRR]) is set, a new count starts. If it is clear, the timer keeps running.

30.3.4 Output Mode

When a timer reaches the reference value selected by DTRR, it can send an output signal on DT*n*OUT. DT*n*OUT can be an active-low pulse or a toggle of the current output, as selected by the DTMR*n*[OM] bit.

30.4 Initialization/Application Information

The general-purpose timer modules typically, but not necessarily, follow this program order:

- The DTMR*n* and DTXMR*n* registers are configured for the desired function and behavior.
  - Count and compare to a reference value stored in the DTRR*n* register
  - Capture the timer value on an edge detected on DT*n*IN
  - Configure DT*n*OUT output mode
  - Increment counter by 1 or by 65,537 (16-bit mode)
  - Enable/disable interrupt or DMA request on counter reference match or capture edge
- The DTMR*n*[CLK] register is configured to select the clock source to be routed to the prescaler.
  - Internal bus clock (can be divided by 1 or 16)
  - DT*n*IN, the maximum value of DT*n*IN is 1/5 of the internal bus clock, as described in the device's electrical characteristics

NOTE

DT*n*IN may not be configured as a clock source when the timer capture mode is selected or indeterminate operation results.

- The 8-bit DTMR*n*[PS] prescaler value is set.
- Using DTMR*n*[RST], counter is cleared and started.
- Timer events are managed with an interrupt service routine, a DMA request, or by a software polling mechanism.

30.4.1 Code Example

The following code provides an example of how to initialize and use DMA Timer0 for counting time-out periods.

```
DTMR0 EQU 0xFC07_0000 ;Timer0 mode register
DTMR1 EQU 0xFC07_4000 ;Timer1 mode register
DTRR0 EQU 0xFC07_0004 ;Timer0 reference register
DTRR1 EQU 0xFC07_4004 ;Timer1 reference register
DTCR0 EQU 0xFC07_0008 ;Timer0 capture register
DTCR1 EQU 0xFC07_4008 ;Timer1 capture register
DTCN0 EQU 0xFC07_000C ;Timer0 counter register
DTCN1 EQU 0xFC07_400C ;Timer1 counter register
DTER0 EQU 0xFC07_0003 ;Timer0 event register
DTER1 EQU 0xFC07_4003 ;Timer1 event register

* TMR0 is defined as: *
*[PS] = 0xFF,     divide clock by 256
```

```
*[CE] = 00       disable capture event output
*[OM] = 0        output=active-low pulse
*[ORRI] = 0,     disable ref. match output
*[FRR] = 1,      restart mode enabled
*[CLK] = 10,     internal bus clock/16
*[RST] = 0,      timer0 disabled
         move.w #0xFF0C,D0
         move.w D0,TMR0
         move.l #0x0000,D0;writing to the timer counter with any
         move.l DO,TCN0 ;value resets it to zero
         move.l #0xAFAF,DO ;set the timer0 reference to be
         move.l #D0,TRR0 ;defined as 0xAFAF
```

The simple example below uses Timer0 to count time-out loops. A time-out occurs when the reference value, 0xAFAF, is reached.

```
timer0_ex
         clr.l DO
         clr.l D1
         clr.l D2
         move.l #0x0000,D0
         move.l D0,TCN0           ;reset the counter to 0x0000
         move.b #0x03,D0          ;writing ones to TER0[REF,CAP]
         move.b D0,TER0           ;clears the event flags
         move.w TMR0,D0           ;save the contents of TMR0 while setting
         bset #0,D0               ;the 0 bit. This enables timer 0 and starts counting
         move.w D0,TMR0           ;load the value back into the register, setting TMR0[RST]
T0_LOOP
         move.b TER0,D1           ;load TER0 and see if
         btst #1,D1               ;TER0[REF] has been set
         beq T0_LOOP
         addi.l #1,D2             ;Increment D2
         cmp.l #5,D2              ;Did D2 reach 5? (i.e. timer ref has timed)
         beq T0_FINISH            ;If so, end timer0 example. Otherwise jump back.
         move.b #0x02,D0          ;writing one to TER0[REF] clears the event flag
         move.b D0,TER0
         jmp T0_LOOP
T0_FINISH
         HALT                     ;End processing. Example is finished
```

30.4.2 Calculating Time-Out Values

Equation 30-1 determines time-out periods for various reference values:

$$\text{Timeout period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}n[\text{PS}] + 1) \times (\text{DTRR}n[\text{REF}] + 1) \qquad \textit{\textbf{Eqn. 30-1}}$$

When calculating time-out periods, add one to the prescaler to simplify calculating, because DTMR*n*[PS] equal to 0x00 yields a prescaler of one, and DTMR*n*[PS] equal to 0xFF yields a prescaler of 256.

For example, if a 133-MHz timer clock is divided by 16, DTMR*n*[PS] equals 0x7F, and the timer is referenced at 0x1FB5B (129,883 decimal), the time-out period is:

$$\text{Timeout period} = \frac{1}{133 \times 10^{6}} \times 16 \times (127 + 1) \times (129883 + 1) = 2.00 \text{ seconds} \qquad \textit{\textbf{Eqn. 30-2}}$$

Chapter 31
DMA Serial Peripheral Interface (DSPI)

31.1 Introduction

This chapter describes the DMA serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

31.1.1 Block Diagram

Figure 31-1 shows a block diagram of the DSPI.

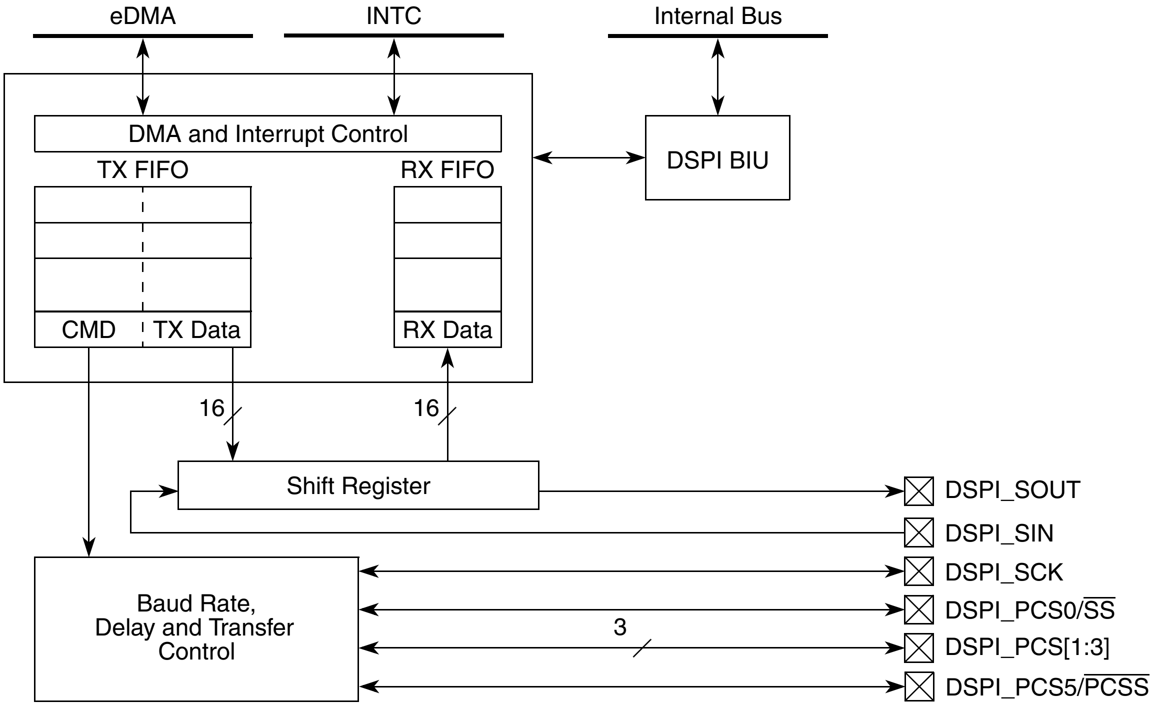


Figure 31-1. DSPI Block Diagram

31.1.2 Overview

The DMA serial peripheral interface (DSPI) block provides a synchronous serial bus for communication between an MCU and an external peripheral device. The DSPI supports up to 32 queued SPI transfers (16 receive and 16 transmit) in the DSPI resident FIFOs eliminating CPU intervention between transfers.

For queued operations, the SPI queues reside in system RAM external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of a DMA controller or through host software.

NOTE

> The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, "Pin Multiplexing and Control") prior to configuring the DSPI.

31.1.3 Features

The DSPI module supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 16 entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - Eight clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    - PCS to SCK delay
    - SCK to PCS delay
    - Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- Five peripheral chip selects, expandable to 32 with external demultiplexer
- Deglitching support for up to seven peripheral chip selects with external demultiplexer
- Two DMA conditions for SPI queues residing in RAM or Flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- Eight interrupt conditions
    - End of queue reached (EOQF)
    - TX FIFO is not full (TFFF)
    - Transfer of current frame complete (TCF)
    - FIFO underflow (slave only, the slave is asked to transfer data when the TX FIFO is empty) (TFUF)
    - RX FIFO is not empty (RFDF)
    - FIFO overflow (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF)

  - FIFO overrun (logical OR of RX overflow and TX underflow interrupts)
  - General DSPI interrupt (logical OR of the seven above conditions)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (DSPI_SCK)

31.1.4 Modes of Operation

The DSPI module has four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode.

Bits in the DSPI_MCR register determine the module-specific modes. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

31.1.4.1 Master Mode

In master mode, the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the DSPI_MCR[MSTR] bit is set. The serial communications clock (DSPI_SCK) is controlled by the master DSPI.

Master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI_CTARs sets the transfer attributes. Transfer attribute control is on a frame by frame basis. See Section 31.4.2, "Serial Peripheral Interface (SPI) Configuration" for more details.

31.1.4.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the DSPI_MCR[MSTR] bit is cleared. A bus master selects the DSPI slave by having the slave's $\overline{\text{DSPI\_SS}}$ signal asserted. In slave mode, the bus master provides DSPI_SCK. The bus master controls all transfer attributes, but clock polarity, clock phase, and numbers of bits to transfer must be configured in the DSPI slave for proper communications.

In slave mode, data transfers MSB first. The LSBFE field of the associated CTAR register is ignored.

31.1.4.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI stops while in module disable mode. The DSPI enters the module disable mode when the DSPI_MCR[MDIS] bit is set. See Section 31.4.7, "Power Saving Features," for more details on the module disable mode.

31.1.4.4 Debug Mode

Debug mode is used for system development and debugging. If the device enters debug mode while the DSPI_MCR[FRZ] bit is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. See Figure 31-12 for a state diagram.

31.2 External Signal Description

31.2.1 Signal Overview

Table 31-1 lists the DSPI signals.

Table 31-1. DSPI Signal Properties

Name	Function			
	Master Mode	**I/O**	**Slave Mode**	**I/O**
DSPI_PCS0/$\overline{\text{SS}}$	Peripheral chip select 0	Output	Slave select	Input
DSPI_PCS[1:3]	Peripheral chip select 1–3	Output	Unused	—
DSPI_PCS5/$\overline{\text{PCSS}}$	Peripheral chip select 5/ Peripheral chip select strobe	Output	Unused	—
DSPI_SIN	Serial data in	Input	Serial data in	Input
DSPI_SOUT	Serial data out	Output	Serial data out	Output
DSPI_SCK	Serial clock	Output	Serial clock	Input

31.2.2 Peripheral Chip Select/Slave Select (DSPI_PCS0/$\overline{\text{SS}}$)

In master mode, the DSPI_PCS0 signal is a peripheral chip select output that selects the slave device to which the current transmission is intended. In slave mode, the DSPI_$\overline{\text{SS}}$ signal is a slave select input signal allowing an SPI master to select the DSPI as the target for transmission.

31.2.3 Peripheral Chip Selects 1–3 (DSPI_PCS[1:3])

The DSPI_PCS[1:3] signals are peripheral chip select output signals in master mode. In slave mode, these signals are not used.

31.2.4 Peripheral Chip Select 5/Peripheral Chip Select Strobe (DSPI_PCS5/$\overline{\text{PCSS}}$)

When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is cleared, DSPI_PCS5 selects the slave device for which the current transfer is intended. DSPI_PCS5 is a peripheral select output signal.

In master mode and when the DSPI_MCR[PCSSE] bit is set, $\overline{\text{DSPI\_PCSS}}$ provides a strobe signal that can be used with an external logic device for deglitching of the PCS signals. $\overline{\text{DSPI\_PCSS}}$ provides the appropriate timing for the decoding of the DSPI_PCS[0:3] signals which prevents glitches from occurring.

DSPI_PCS5/$\overline{\text{PCSS}}$ is not used in slave mode.

31.2.5 Serial Input (DSPI_SIN)

DSPI_SIN is a serial data input signal.

31.2.6 Serial Output (DSPI_SOUT)

DSPI_SOUT is a serial data output signal.

31.2.7 Serial Clock (DSPI_SCK)

DSPI_SCK is a serial communication clock signal. In master mode, DSPI generates DSPI_SCK. In slave mode, DSPI_SCK is an input from an external bus master.

31.3 Memory Map/Register Definition

Table 31-2 shows the DSPI memory map.

Table 31-2. DSPI Module Memory Map

Address	Register	Width	Access	Reset Value	Section/Page
0xFC05_C000	DSPI module configuration register (DSPI_MCR)	32	R/W	0x0000_4001	31.3.1/31-5
0xFC05_C008	DSPI transfer count register (DSPI_TCR)	32	R/W	0x0000_0000	31.3.2/31-8
0xFC05_C00C + (*n* × 0x04)	DSPI clock and transfer attributes registers (DSPI_CTAR*n*), *n*=0:7	32	R/W	0x7800_0000	31.3.3/31-8
0xFC05_C02C	DSPI status register (DSPI_SR)	32	R/W	0x0000_0000	31.3.4/31-14
0xFC05_C030	DSPI DMA/interrupt request select and enable register (DSPI_RSER)	32	R/W	0x0000_0000	31.3.5/31-16
0xFC05_C034	DSPI push TX FIFO register (DSPI_PUSHR)	32	R/W	0x0000_0000	31.3.6/31-17
0xFC05_C038	DSPI pop RX FIFO register (DSPI_POPR)	32	R	0x0000_0000	31.3.7/31-19
0xFC05_C03C + (*n* × 0x04)	DSPI transmit FIFO registers (DSPI_TXFR*n*), *n*=0:15	32	R	0x0000_0000	31.3.8/31-19
0xFC05_C07C + (*n* × 0x04)	DSPI receive FIFO registers (DSPI_RXFR*n*), *n*=0:15	32	R	0x0000_0000	31.3.9/31-20

31.3.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits that configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR may be changed while the DSPI is running.

NOTE

The DSPI_MCR[MDIS] bit is set at reset.

Address: 0xFC05_C000 (DSPI_MCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MSTR	CONT_ SCKE	DCONF		FRZ	MTFE	PCS SE	RO OE	PCS IS7	PCS IS6	PCS IS5	PCS IS4	PCS IS3	PCS IS2	PCS IS1	PCS IS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	MDIS	DIS_ TXF	DIS_ RXF	0	0	SMPL_PT		0	0	0	0	0	0	0	HALT
W					CLR_ TXF	CLR_ RXF										
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 31-2. DSPI Module Configuration Register (DSPI_MCR)

Table 31-3. DSPI_MCR Field Descriptions

Field	Description
31 MSTR	Master/slave mode select. Configures the DSPI for master mode or slave mode. **Note:** This bit's value must only be changed when the DSPI_MCR[HALT] bit is set. Otherwise, improper operation may occur. 0 Slave mode 1 Master mode
30 CONT_ SCKE	Continuous SCK enable. Enables the serial communication clock (DSPI_SCK) to run continuously. See Section 31.4.5, "Continuous Serial Communications Clock," for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
29–28 DCONF	DSPI configuration. Selects between the different configurations of the DSPI. 00 SPI 01 Reserved 10 Reserved 11 Reserved **Note:** All values except 00 are reserved. This field must be configured for SPI mode for the DSPI module to operate correctly.
27 FRZ	Freeze. Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers 1 Halt serial transfers
26 MTFE	Modified timing format enable. Enables a modified transfer format to be used. See Section 31.4.4.4, "Modified SPI Transfer Format (MTFE = 1, CPHA = 1)," for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled

Table 31-3. DSPI_MCR Field Descriptions (continued)

Field	Description
25 PCSSE	Peripheral chip select strobe enable. Selects between the DSPI_PCS5 and $\overline{\text{DSPI\_PCSS}}$ functions on the DSPI_PCS5/$\overline{\text{PCSS}}$ signal. See Section 31.4.3.5, “Peripheral Chip Select Strobe Enable (PCSS),” for more information. 0 DSPI_PCS5/$\overline{\text{PCSS}}$ is used as the peripheral chip select 5 signal 1 DSPI_PCS5/$\overline{\text{PCSS}}$ is used as an active-low PCS strobe signal
24 ROOE	Receive FIFO overflow overwrite enable. Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is set, incoming data is shifted into the shift register. If the ROOE bit is cleared, incoming data is ignored. See Section 31.4.6.6, “Receive FIFO Overflow Interrupt Request (RFOF),” for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
23–16 PCSIS*n*	Peripheral chip select inactive state. Determines the inactive state of the DSPI_PCS*n* signal. 0 The inactive state of DSPI_PCS*n* is low 1 The inactive state of DSPI_PCS*n* is high **Note:** DSPI_PCS7, DSPI_PCS6, and DSPI_PCS4 are not implemented on this device. Therefore, these corresponding bits are reserved. **Note:** DSPI_PCS0/$\overline{\text{SS}}$ must be configured as inactive high for slave mode operation.
15	Reserved, must be cleared.
14 MDIS	Module disable. Allows the clock to be stopped to non-memory mapped logic in DSPI effectively putting DSPI in a software controlled power-saving state. See Section 31.4.7, “Power Saving Features,” for more information. This bit is set at reset. 0 Enable DSPI clocks 1 Allow external logic to disable DSPI clocks
13 DIS_TXF	Disable transmit FIFO. When the TX FIFO is disabled, transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 31.4.2.3, “FIFO Disable Operation,” for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
12 DIS_RXF	Disable receive FIFO. When the RX FIFO is disabled, receive part of the DSPI operates as a simplified double-buffered SPI. See Section 31.4.2.3, “FIFO Disable Operation for details.” 0 RX FIFO is enabled 1 RX FIFO is disabled
11 CLR_TXF	Clear TX FIFO. Flushes the TX FIFO. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO counter 1 Clear the TX FIFO counter **Note:** When the respective FIFO is disabled, this bit does has no effect.
10 CLR_RXF	Clear RX FIFO. Flushes the RX FIFO. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO counter 1 Clear the RX FIFO counter **Note:** When the respective FIFO is disabled, this bit does has no effect.
9–8 SMPL_PT	Sample point. Allows host software to select when the DSPI master samples SIN in modified transfer format. Figure 31-17 shows where the master can sample the SIN pin. 00 0 system clocks between DSPI_SCK edge and DSPI_SIN sample 01 1 system clock between DSPI_SCK edge and DSPI_SIN sample 10 2 system clocks between DSPI_SCK edge and DSPI_SIN sample 11 Reserved

Table 31-3. DSPI_MCR Field Descriptions (continued)

Field	Description
7–1	Reserved, must be cleared.
0 HALT	Halt. Starts and stops DSPI transfers. See Section 31.4.1, “Start and Stop of DSPI Transfers,” for details on the operation of this bit. 0 Start transfers 1 Stop transfers

31.3.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write to the DSPI_TCR while the DSPI is running.

Address: 0xFC05_C008 (DSPI_TCR)

Access: User 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
R	SPI_TCNT																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-3. DSPI Transfer Count Register (DSPI_TCR)

Table 31-4. DSPI_TCR Field Descriptions

Field	Description
31–16 SPI_TCNT	SPI transfer counter. Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of an SPI frame transmits. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to 0 at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter wraps around. Incrementing the counter past 65535 resets the counter to 0.
15–0	Reserved, must be cleared

31.3.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR*n*)

DSPI modules each contain eight clock and transfer attribute registers (DSPI_CTAR*n*) used to define different transfer attribute configurations. Each DSPI_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB/LSB first

DSPI_CTARs support compatibility with the QSPI module in the ColdFire family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPI_CTAR that contains the transfer’s attributes. Do not write to the DSPI_CTARs while the DSPI is running.

In master mode, the DSPI_CTAR*n* registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. When DSPI is configured as an SPI master, the DSPI_PUSHR[CTAS] field in the command portion of the TX FIFO entry selects which of the DSPI_CTAR registers is used on a per-frame basis.

In slave mode, a subset of the bit fields in the DSPI_CTAR0 registers sets the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

Address: 0xFC05_C00C (DSPI_CTAR0)
0xFC05_C010 (DSPI_CTAR1)
0xFC05_C014 (DSPI_CTAR2)
0xFC05_C018 (DSPI_CTAR3)
0xFC05_C01C (DSPI_CTAR4)
0xFC05_C020 (DSPI_CTAR5)
0xFC05_C024 (DSPI_CTAR6)
0xFC05_C028 (DSPI_CTAR7)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DBR	FMSZ				CPOL	CPHA	LSB FE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-4. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR*n*)

Table 31-5. DSPI_CTAR*n* Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31
DBR</td><td>Double baud rate. The DBR bit doubles the effective baud rate of the serial communications clock (SCK). This field is only used in master mode. It effectively halves the baud rate division ratio supporting faster frequencies and odd division ratios for the serial communications clock (SCK). When the DBR bit is set, the duty cycle of the serial communications clock (SCK) depends on the value in the baud rate prescaler and the clock phase bit as listed below. See the BR field below and Section 31.4.3.1, “Baud Rate Generator” for details on how to compute the baud rate. If the overall baud rate is divided by two or three of the system clock, the continuous SCK enable or the modified timing format enable bits must not be set.
0 The baud rate is computed normally with a 50/50 duty cycle
1 Baud rate is doubled with the duty cycle depending on the baud rate prescaler
<table>
<tr><th>DBR</th><th>CPHA</th><th>PBR</th><th>SCK Duty Cycle</th></tr>
<tr><td>0</td><td>any</td><td>any</td><td>50/50</td></tr>
<tr><td>1</td><td>0</td><td>00</td><td>50/50</td></tr>
<tr><td>1</td><td>0</td><td>01</td><td>33/66</td></tr>
<tr><td>1</td><td>0</td><td>10</td><td>40/60</td></tr>
<tr><td>1</td><td>0</td><td>11</td><td>43/57</td></tr>
<tr><td>1</td><td>1</td><td>00</td><td>50/50</td></tr>
<tr><td>1</td><td>1</td><td>01</td><td>66/33</td></tr>
<tr><td>1</td><td>1</td><td>10</td><td>60/40</td></tr>
<tr><td>1</td><td>1</td><td>11</td><td>57/43</td></tr>
</table>
</td></tr>
<tr><td>30–27
FMSZ</td><td>Frame size. Selects the number of bits transferred per frame. The FMSZ field is used in master mode and slave mode. The table below lists the frame sizes.
<table>
<tr><th>FMSZ</th><th>Framesize</th><th>FMSZ</th><th>Framesize</th></tr>
<tr><td>0000</td><td>Reserved</td><td>1000</td><td>9</td></tr>
<tr><td>0001</td><td>Reserved</td><td>1001</td><td>10</td></tr>
<tr><td>0010</td><td>Reserved</td><td>1010</td><td>11</td></tr>
<tr><td>0011</td><td>4</td><td>1011</td><td>12</td></tr>
<tr><td>0100</td><td>5</td><td>1100</td><td>13</td></tr>
<tr><td>0101</td><td>6</td><td>1101</td><td>14</td></tr>
<tr><td>0110</td><td>7</td><td>1110</td><td>15</td></tr>
<tr><td>0111</td><td>8</td><td>1111</td><td>16</td></tr>
</table>
</td></tr>
</table>

Table 31-5. DSPI_CTAR*n* Field Description (continued)

Field	Description
26 CPOL	Clock polarity. Selects the inactive state of the serial communications clock (DSPI_SCK). This bit is used in master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the continuous selection format is selected (CONT or DCONT is set), switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. For more information on continuous selection format, refer to Section 31.4.4.5, "Continuous Selection Format." 0 The inactive state value of DSPI_SCK is low 1 The inactive state value of DSPI_SCK is high
25 CPHA	Clock phase. Selects which edge of DSPI_SCK causes data to change and which edge causes data to be captured. This bit is used in master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings. **Note:** When the continuous selection format is selected (CONT or DCONT is set), switching between clock phases without stopping the DSPI can cause errors in the transfer. 0 Data is captured on the leading edge of DSPI_SCK and changed on the following edge 1 Data is changed on the leading edge of DSPI_SCK and captured on the following edge
24 LSBFE	LSB first enable. Selects if the LSB or MSB of the frame is transferred first. This bit is only used in master mode. 0 Data is transferred MSB first 1 Data is transferred LSB first
23–22 PCSSCK	PCS to SCK delay prescaler. Selects the prescaler value for the delay between assertion of DSPI_PCS and the first edge of the DSPI_SCK. This field is only used in master mode. **Note:** When the continuous selection format is selected (CONT or DCONT is set), switching the PCS to SCK delay prescaler without stopping the DSPI can cause errors in the transfer. **Note:** See Section 31.4.3.2, "PCS to SCK Delay (tCSC)," for details on calculating the PCS to SCK delay. 00 1 clock DSPI_PCS to DSPI_SCK delay prescaler 01 3 clock DSPI_PCS to DSPI_SCK delay prescaler 10 5 clock DSPI_PCS to DSPI_SCK delay prescaler 11 7 clock DSPI_PCS to DSPI_SCK delay prescaler
21–20 PASC	After SCK delay prescaler. Selects the prescaler value for the delay between the last edge of DSPI_SCK and the negation of DSPI_PCS. This field is only used in master mode. The ASC field description in Table 31-5 explains how to compute the after SCK delay. 00 1 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler 01 3 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler 10 5 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler 11 7 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler
19–18 PDT	Delay after transfer prescaler. The PDT field selects the prescaler value for the delay between the negation of the DSPI_PCS signal at the end of a frame and the assertion of DSPI_PCS at the beginning of the next frame. The PDT field is only used in master mode. The DT field description in Table 31-5 explains how to compute the delay after transfer. 00 1 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler 01 3 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler 10 5 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler 11 7 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler

Table 31-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>17–16
PBR</td><td>Baud rate prescaler. Selects the prescaler value for the baud rate. This field is only used in master mode. The baud rate is the frequency of the serial communications clock (DSPI_SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The description for Section 31.4.3.1, “Baud Rate Generator” details how to compute the baud rate.
00 2 clock prescaler to divide system clock
01 3 clock prescaler to divide system clock
10 5 clock prescaler to divide system clock
11 7 clock prescaler to divide system clock</td></tr>
<tr><td>15–12
CSSCK</td><td>PCS to SCK delay scaler. Selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK delay is the delay between the assertion of DSPI_PCS and the first edge of the DSPI_SCK. The table below lists the scaler values.
<b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching the PCS to SCK delay prescaler without stopping the DSPI can cause errors in the transfer.
<table>
<tr><th>CSSCK</th><th>PCS to SCK Delay Scaler Value</th><th>CSSCK</th><th>PCS to SCK Delay Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
<b>Note:</b> See Section 31.4.3.2, “PCS to SCK Delay (tCSC),” for details on calculating the PCS to SCK delay.</td></tr>
</table>

Table 31-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>11–8
ASC</td><td>After SCK delay scaler. Selects the scaler value for the after SCK delay. This field is only used in master mode. The after SCK delay is the delay between the last edge of DSPI_SCK and the negation of DSPI_PCS. The table below lists the scaler values.
<table>
<tr><th>ASC</th><th>After SCK Delay Scaler Value</th><th>ASC</th><th>After SCK Delay Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
<b>Note:</b> See Section 31.4.3.3, “After SCK Delay (tASC),” for more details on calculating the after SCK delay.</td></tr>
<tr><td>7–4
DT</td><td>Delay after transfer scaler. The DT field selects the delay after transfer scaler. This field is only used in master mode. The delay after transfer is the time between the negation of the DSPI_PCS signal at the end of a frame and the assertion of DSPI_PCS at the beginning of the next frame. The table below lists the scaler values.
<table>
<tr><th>DT</th><th>Delay after Transfer Scaler Value</th><th>DT</th><th>Delay after Transfer Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
<b>Note:</b> See Section 31.4.3.4, “Delay after Transfer (tDT),” for more details on calculating the delay after transfer.</td></tr>
</table>

Table 31-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–0
BR</td><td>Baud rate scaler. Selects the scaler value for the baud rate. This field is only used in master mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the DSPI_SCK. The table below lists the baud rate scaler values.
<table>
<tr><th>BR</th><th>Baud Rate Scaler Value</th><th>BR</th><th>Baud Rate Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>256</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>512</td></tr>
<tr><td>0010</td><td>6</td><td>1010</td><td>1024</td></tr>
<tr><td>0011</td><td>8</td><td>1011</td><td>2048</td></tr>
<tr><td>0100</td><td>16</td><td>1100</td><td>4096</td></tr>
<tr><td>0101</td><td>32</td><td>1101</td><td>8192</td></tr>
<tr><td>0110</td><td>64</td><td>1110</td><td>16384</td></tr>
<tr><td>0111</td><td>128</td><td>1111</td><td>32768</td></tr>
</table>
<b>Note:</b> See Section 31.4.3.1, “Baud Rate Generator,” for more details on calculating the baud rate.</td></tr>
</table>

31.3.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a 1 to it. Writing a 0 to a flag bit has no effect.

Address
: 0xFC05_C02C (DSPI_SR)

Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-5. DSPI Status Register (DSPI_SR)

Table 31-6. DSPI_SR Field Descriptions

Field	Description
31 TCF	Transfer complete flag. Indicates all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the t_{ASC} delay starts. Refer to Section 31.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete
30 TXRXS	TX and RX status. Reflects the status of the DSPI. See Section 31.4.1, “Start and Stop of DSPI Transfers” for information on what causes this bit to be cleared or set. 0 TX and RX operations are disabled (DSPI is in stopped state) 1 TX and RX operations are enabled (DSPI is in running state)
29	Reserved, must be cleared.
28 EOQF	End of queue flag. Indicates transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the t_{ASC} delay starts. Refer to Section 31.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” for details. The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing SPI command 1 EOQ bit is set in the executing SPI command **Note:** EOQF does not function in slave mode.
27 TFUF	Transmit FIFO underflow flag. Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
26	Reserved, must be cleared.
25 TFFF	Transmit FIFO fill flag. Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. Therefore, this bit is set after DSPI_MCR[MDIS] is cleared after a reset. The TFFF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full
24–20	Reserved, must be cleared.
19 RFOF	Receive FIFO overflow flag. Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
18	Reserved, must be cleared.
17 RFDF	Receive FIFO drain flag. Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty **Note:** In the interrupt service routine, RFDF must be cleared only after the DSPI_POPR register is read.
16	Reserved, must be cleared.

Table 31-6. DSPI_SR Field Descriptions (continued)

Field	Description
15–12 TXCTR	TX FIFO counter. Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
11–8 TXNXTPTR	Transmit next pointer. Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 31.4.2.4, "TX FIFO Buffering Mechanism," for more details.
7–4 RXCTR	RX FIFO counter. Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI_POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the t_{ASC} delay starts. Refer to Section 31.4.4.1, "Classic SPI Transfer Format (CPHA = 0)" for details.
3–0 POPNXTPTR	Pop next pointer. Contains a pointer to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 31.4.2.5, "RX FIFO Buffering Mechanism" for more details.

31.3.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER serves two purposes. It enables flag bits in the DSPI_SR to generate DMA requests or interrupt requests. The DSPI_RSER also selects the type of request to be generated. See the individual bit descriptions for information on the types of requests the bits support. Do not write to the DSPI_RSER while the DSPI is running.

Address 0xFC05_C030 (DSPI_RSER)
:

Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-6. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 31-7. DSPI_RSER Field Descriptions

Field	Description
31 TCF_RE	Transmission complete request enable. Enables DSPI_SR[TCF] flag to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
30–29	Reserved, must be cleared.

Table 31-7. DSPI_RSER Field Descriptions (continued)

Field	Description
28 EOQF_RE	DSPI finished request enable. Enables the DSPI_SR[EOQF] flag to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
27 TFUF_RE	Transmit FIFO underflow request enable. Enables the DSPI_SR[TFUF] flag to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
26	Reserved, must be cleared.
25 TFFF_RE	Transmit FIFO fill request enable. Enables the DSPI_SR[TFFF] flag to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt or DMA requests are disabled 1 TFFF interrupt or DMA requests are enabled
24 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the DSPI_SR[TFFF] flag bit and the DSPI_RSER[TFFF_RE] bit are set, this bit selects between generating an interrupt request or a DMA request. 0 TFFF flag generates interrupt requests 1 TFFF flag generates DMA requests
23–20	Reserved, must be cleared.
19 RFOF_RE	Receive FIFO overflow request enable. Enables the DSPI_SR[RFOF] flag to generate an interrupt request. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
18	Reserved, must be cleared.
17 RFDF_RE	Receive FIFO drain request enable. Enables the DSPI_SR[RFDF] flag to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt or DMA requests are disabled 1 RFDF interrupt or DMA requests are enabled
16 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the DSPI_SR[RFDF] flag bit and the DSPI_RSER[RFDF_RE] bit are set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF flag generates interrupt requests 1 RFDF flag generates DMA requests
15–0	Reserved, must be cleared.

31.3.6 DSPI Push Transmit FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides a means to write to the TX FIFO. SPI commands and data written to this register is transferred to the TX FIFO. See Section 31.4.2.4, “TX FIFO Buffering Mechanism,” for more information. Write accesses of 8- or 16-bits to the DSPI_PUSHR transfer 32 bits to the TX FIFO.

NOTE

Only the TXDATA field is used for DSPI slaves.

Address: 0xFC05_C034 (DSPI_PUSHR) Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CONT	CTAS			EOQ	CT CNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-7. DSPI Push Transmit FIFO Register (DSPI_PUSHR)

Table 31-8. DSPI_PUSHR Field Descriptions

Field	Description
31 CONT	Continuous peripheral chip select enable. Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 31.4.4.5, “Continuous Selection Format,” for more information. 0 Return DSPI_PCS*n* signals to their inactive state between transfers 1 Keep DSPI_PCS*n* signals asserted between transfers
30–28 CTAS	Clock and transfer attributes select. Selects which of the DSPI_CTAR*n* registers is used to set the transfer attributes for the associated SPI frame. This field is used only in SPI master mode. In SPI slave mode, DSPI_CTAR0 is used instead. 000 DSPI_CTAR0 001 DSPI_CTAR1 010 DSPI_CTAR2 011 DSPI_CTAR3 100 DSPI_CTAR4 101 DSPI_CTAR5 110 DSPI_CTAR6 111 DSPI_CTAR7
27 EOQ	End of queue. Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the DSPI_SR[EOQF] bit is set. This bit is used only in SPI master mode. 0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer
26 CTCNT	Clear SPI_TCNT. Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the DSPI_TCR[SPI_TCNT] field. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. This bit is used only in SPI master mode. 0 Do not clear DSPI_TCR[SPI_TCNT] field 1 Clear DSPI_TCR[SPI_TCNT] field
25–24	Reserved, must be cleared.

Table 31-8. DSPI_PUSHR Field Descriptions (continued)

Field	Description
23–16 PCS*n*	Peripheral chip select *n*. Selects which DSPI_PCS*n* signals are asserted for the transfer. This bit is used only in SPI master mode. 0 Negate the DSPI_PCS*n* signal 1 Assert the DSPI_PCS*n* signal **Note:** DSPI_PCS7, DSPI_PCS6, and DSPI_PCS4 are not implemented on this device. Therefore, these corresponding bits are reserved.
15–0 TXDATA	Transmit data. Holds SPI data to be transferred according to the associated SPI command. **Note:** TXDATA is used in slave mode.

31.3.7 DSPI Pop Receive FIFO Register (DSPI_POPR)

The DSPI_POPR provides a means to read the RX FIFO. See Section 31.4.2.5, “RX FIFO Buffering Mechanism” for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI_POPR read from the RX FIFO and update the counter and pointer.

Address: 0xFC05_C038 (DSPI_POPR)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RXDATA															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-8. DSPI Pop Receive FIFO Register (DSPI_POPR)

Table 31-9. DSPI_POPR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 RXDATA	Received data. Contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (DSPI_SR[POPNXTPTR]).

31.3.8 DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR*n*)

The DSPI_TXFR*n* registers provide visibility into TX FIFO for debugging purposes. Each register is an entry in TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFR*n* registers does not alter the state of TX FIFO. The 16-entry deep FIFO is implemented with 16 registers, DSPI_TXFR0–15.

Address: 0xFC05_C03C (DSPI_TXFR0) 0xFC05_C05C (DSPI_TXFR8)
0xFC05_C040 (DSPI_TXFR1) 0xFC05_C060 (DSPI_TXFR9)
0xFC05_C044 (DSPI_TXFR2) 0xFC05_C064 (DSPI_TXFR10)
0xFC05_C048 (DSPI_TXFR3) 0xFC05_C068 (DSPI_TXFR11)
0xFC05_C04C (DSPI_TXFR4) 0xFC05_C06C (DSPI_TXFR12)
0xFC05_C050 (DSPI_TXFR5) 0xFC05_C070 (DSPI_TXFR13)
0xFC05_C054 (DSPI_TXFR6) 0xFC05_C074 (DSPI_TXFR14)
0xFC05_C058 (DSPI_TXFR7) 0xFC05_C078 (DSPI_TXFR15)

Access: User 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
R	TXCMD																TXDATA															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-9. DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR*n*)

Table 31-10. DSPI_TXFR*n* Field Descriptions

Field	Description
31–16 TXCMD	Transmit command. Contains the command that sets the transfer attributes for the SPI data. See Section 31.3.6, “DSPI Push Transmit FIFO Register (DSPI_PUSHR),” for details on the command field.
15–0 TXDATA	Transmit data. Contains the SPI data to be shifted out.

31.3.9 DSPI Receive FIFO Registers 0–15 (DSPI_RXFR*n*)

The DSPI_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFR*n* registers does not alter the state of the RX FIFO. The device uses 16 registers to implement the RX FIFO; DSPI_RXFR0–15 are used.

Address: 0xFC05_C07C (DSPI_RXFR0) 0xFC05_C09C (DSPI_RXFR8)
0xFC05_C080 (DSPI_RXFR1) 0xFC05_C0A0 (DSPI_RXFR9)
0xFC05_C084 (DSPI_RXFR2) 0xFC05_C0A4 (DSPI_RXFR10)
0xFC05_C088 (DSPI_RXFR3) 0xFC05_C0A8 (DSPI_RXFR11)
0xFC05_C08C (DSPI_RXFR4) 0xFC05_C0AC (DSPI_RXFR12)
0xFC05_C090 (DSPI_RXFR5) 0xFC05_C0B0 (DSPI_RXFR13)
0xFC05_C094 (DSPI_RXFR6) 0xFC05_C0B4 (DSPI_RXFR14)
0xFC05_C098 (DSPI_RXFR7) 0xFC05_C0B8 (DSPI_RXFR15)

Access: User 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RXDATA															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-10. DSPI Receive FIFO Registers (DSPI_RXFR*n*)

Table 31-11. DSPI_RXFR*n* Field Description

Field	Description
31–16	Reserved, must be cleared.
15–0 RXDATA	Receive data. Contains the received SPI data.

31.4 Functional Description

The DSPI supports full-duplex, synchronous serial communications between the MCU and external peripheral devices. The DSPI supports up to 32 queued SPI transfers at once (16 transmit and 16 receive) in the DSPI resident FIFOs, thereby eliminating CPU intervention between transfers.

The DSPI_CTAR*n* registers hold clock and transfer attributes. The SPI configuration can select which CTAR to use on a frame by frame basis by setting the DSPI_PUSHR[CTAS] field. See Section 31.3.3, "DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTARn)," for information on DSPI_CTAR*n* fields.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT and SIN signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data exchanged between the master and the slave; the data that was in the master's shift register is now in the shift register of the slave and vice versa. At the end of a transfer, the DSPI_SR[TCF] bit is set to indicate a completed transfer. Figure 31-11 illustrates how master and slave data is exchanged.

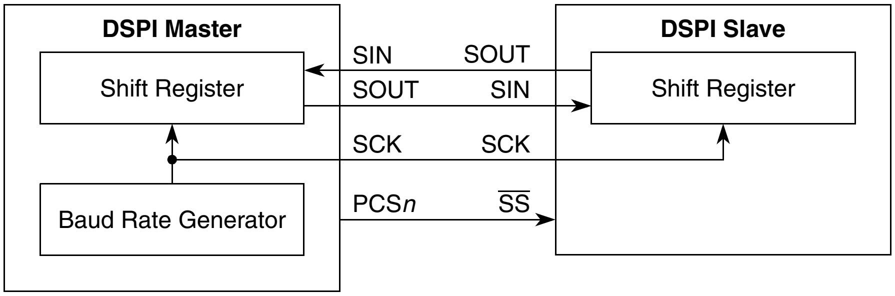


Figure 31-11. SPI Serial Protocol Overview

The DSPI has five peripheral chip select (DSPI_PCS*n*) signals that select which of the slaves to communicate with.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in Section 31.4.4, "Transfer Formats." The transfer rate and delay settings are described in section Section 31.4.3, "DSPI Baud Rate and Clock Delay Generation."

See Section 31.4.7, "Power Saving Features" for information on the power-saving features of the DSPI.

31.4.1 Start and Stop of DSPI Transfers

The DSPI has two operating states; stopped and running. The default state of the DSPI is stopped. In the stopped state, no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The stopped state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. Master/slave mode must only be changed when the DSPI is halted (DSPI_MCR[HALT] is set). The DSPI_SR[TXRXS] bit is cleared in this state. In the running state, serial transfers take place. The DSPI_SR[TXRXS] bit is set in the running state. Figure 31-12 shows a state diagram of the start and stop mechanism. The transitions are described in Table 31-12.

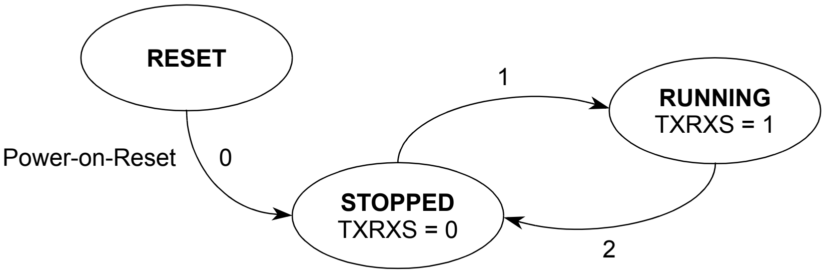


Figure 31-12. DSPI Start and Stop State Diagram

Table 31-12. State Transitions for Start and Stop of DSPI Transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI is started (DSPI transitions to running) when all of the following conditions are true: • EOQF bit is clear • Debug mode is unselected or the FRZ bit is clear • HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from running to stopped) after the current frame for any one of the following conditions: • EOQF bit is set • Debug mode is selected and the FRZ bit is set • HALT bit is set

State transitions from running to stopped occur on the next frame boundary if a transfer is in progress or on the next system clock cycle if no transfers are in progress.

31.4.2 Serial Peripheral Interface (SPI) Configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The SPI frames can be from 4–16 bits long. The data transmitted can come from queues stored in RAM external to the DSPI. Host software or the eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or the eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI. The FIFO buffer operations are described in Section 31.4.2.4, “TX FIFO Buffering Mechanism,” and Section 31.4.2.5, “RX FIFO Buffering Mechanism.” The interrupt and DMA request conditions are described in Section 31.4.6, “Interrupts/DMA Requests.”

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for both modes. In master mode, the DSPI initiates and controls the transfer according to the SPI command field of the TX FIFO entry. In slave mode, the DSPI only responds to transfers initiated by a bus master external to the DSPI, and the SPI command field of the TX FIFO entry is ignored. For information on switching between master and slave modes see Section 31.5.2, “Switching Master and Slave Mode.”

31.4.2.1 Master Mode

In master mode, the DSPI initiates the serial transfers by controlling the serial communications clock (DSPI_SCK) and the peripheral chip select (DSPI_PCS*n*) signals. The SPI command field in the executing TX FIFO entry determines which DSPI_CTAR*n* register sets the transfer attributes and which DSPI_PCS*n* signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See Section 31.3.6, "DSPI Push Transmit FIFO Register (DSPI_PUSHR)," for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (DSPI_SOUT) pin. In master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

31.4.2.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase, and frame size must be set for successful communication with an SPI master. The slave mode transfer attributes are set in the DSPI_CTAR0 register.

31.4.2.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX or RX FIFOs. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The FIFOs are disabled separately; setting the DSPI_MCR[DIS_TXF] bit disables the TX FIFO, and setting the DSPI_MCR[DIS_RXF] bit disables the RX FIFO.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR. When the TX FIFO is disabled, DSPI_SR[TFFF, TFUF, and TXCTR] fields behave as if there is a one-entry FIFO, but the contents of DSPI_TXFRs and TXNXTPTR are undefined. Likewise, when RX FIFO is disabled, DSPI_SR[RFDF, RFOF, and RXCTR] fields behave as if there is a one-entry FIFO, but the contents of DSPI_RXFRs and POPNXTPTR are undefined.

The TX and RX FIFOs should be disabled only if the application's operating mode requires FIFO to be disabled. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported and may result in incorrect results.

NOTE

When the FIFOs are disabled, the respective DSPI_MCR[CLR_TXF, CLR_RXF] bits have no effect.

31.4.2.4 TX FIFO Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds 16 entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI_PUSHR). For more information on DSPI_PUSHR, refer to Section 31.3.6, "DSPI Push Transmit FIFO Register (DSPI_PUSHR)." TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPI_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data transfers into the shift register from the TX FIFO. For more information on DSPI_SR, refer to Section 31.3.4, “DSPI Status Register (DSPI_SR).”

The DSPI_SR[TXNXTPTR] field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means DSPI_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field increments every time SPI data transfers from TX FIFO to shift register.

31.4.2.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR register. When the TX FIFO is not full, the TX FIFO fill flag, DSPI_SR[TFFF], is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI_PUSHR is complete. Host software writing a 1 to the DSPI_SR[TFFF] bit can also clear the TFFF bit. The TFFF can generate a DMA request or an interrupt request. See Section 31.4.6.2, “Transmit FIFO Fill Interrupt or DMA Request (TFFF),” for details.

The DSPI ignores attempts to push data to a full TX FIFO; in other words, the state of the TX FIFO is unchanged and no error condition is indicated.

31.4.2.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter decrements by one. At the end of a transfer, the DSPI_SR[TCF] bit is set to indicate completion of a transfer. The TX FIFO is flushed by writing a 1 to the DSPI_MCR[CLR_TXF] bit.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the slave’s transmit FIFO underflow flag, DSPI_SR[TFUF], is set. See Section 31.4.6.4, “Transmit FIFO Underflow Interrupt Request (TFUF),”for details.

31.4.2.5 RX FIFO Buffering Mechanism

The RX FIFO functions as a buffer for data received on the DSPI_SIN pin. The RX FIFO holds 16 received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPI_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO. For more information on the DSPI_POPR, refer to Section 31.3.7, “DSPI Pop Receive FIFO Register (DSPI_POPR).”

The RX FIFO counter field, DSPI_SR[RXCTR], indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The DSPI_SR[POPNXTPTR] field points to the RX FIFO entry returned when the DSPI_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPI_RXFR0. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that is returned when DSPI_POPR is read. The POPNXTPTR field increments every time the DSPI_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

31.4.2.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO, the RX FIFO counter increments by one.

If the RX FIFO and shift register are full and a transfer is initiated, the DSPI_SR[RFOF] bit is asserted indicating an overflow condition. Depending on the state of the DSPI_MCR[ROOE] bit, data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is set, incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

31.4.2.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI_POPR. For more information on DSPI_POPR, refer to Section 31.3.7, "DSPI Pop Receive FIFO Register (DSPI_POPR)." A read of the DSPI_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, and the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

When the RX FIFO is not empty, the RX FIFO drain flag, DSPI_SR[RFDF], is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPI_POPR is complete. Alternatively, the RFDF bit can be cleared by software writing a 1 to it.

31.4.3 DSPI Baud Rate and Clock Delay Generation

The DSPI_SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate. Figure 31-13 shows conceptually how the DSPI_SCK signal is generated.

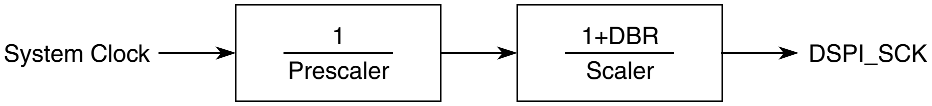


Figure 31-13. Communications Clock Prescalers and Scalers

31.4.3.1 Baud Rate Generator

The baud rate is the frequency of the serial communication clock (DSPI_SCK). The system clock is divided by a baud rate prescaler (defined by DSPI_CTAR*n*[PBR]) and baud rate scaler (defined by DSPI_CTAR*n*[BR]) to produce DSPI_SCK with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI_CTAR*n* select the frequency of DSPI_SCK using the following formula:

$$\text{SCK baud rate} = \frac{f_{SYS/2}}{\text{PBR Prescaler Value}} \times \frac{1 + \text{DBR}}{\text{BR Scaler Value}} \qquad \textit{Eqn. 31-1}$$

Table 31-13 shows an example of a computed baud rate.

Table 31-13. Baud Rate Computation Example

$f_{SYS/2}$	PBR	Prescaler Value	BR	Scaler Value	DBR Value	Baud Rate
100 MHz	00	2	0000	2	0	25 Mb/s
20 MHz	00	2	0000	2	1	10 Mb/s

31.4.3.2 PCS to SCK Delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of DSPI_PCS signal to the first DSPI_SCK edge. See Figure 31-15 for an illustration of the PCS to SCK delay. The DSPI_CTAR*n*[PCSSCK, CSSCK] fields select the PCS to SCK delay, and the relationship is expressed by the following:

$$t_{CSC} = \frac{1}{f_{SYS/2}} \times PCSSCK \times CSSCK \qquad \textbf{\textit{Eqn. 31-2}}$$

Table 31-14 shows an example of the computed PCS to SCK delay.

Table 31-14. PCS to SCK Delay Computation Example

PCSSCK	Prescaler Value	CSSCK	Scaler Value	$f_{SYS/2}$	PCS to SCK Delay
01	3	0100	32	100 MHz	0.96 µs

31.4.3.3 After SCK Delay (t_{ASC})

The after SCK delay is the length of time between the last edge of DSPI_SCK and negation of DSPI_PCS. See Figure 31-15 and Figure 31-16 for illustrations of the after SCK delay. The DSPI_CTAR*n*[PASC, ASC] fields select the after SCK delay. The relationship between these variables is given in the following:

$$t_{ASC} = \frac{1}{f_{SYS/2}} \times PASC \times ASC \qquad \textbf{\textit{Eqn. 31-3}}$$

Table 31-15 shows an example of the computed after SCK delay.

Table 31-15. After SCK Delay Computation Example

PASC	Prescaler Value	ASC	Scaler Value	$f_{SYS/2}$	After SCK Delay
01	3	0100	32	100 MHz	0.96 us

31.4.3.4 Delay after Transfer (t_{DT})

The delay after transfer is the length of time between negation of DSPI_PCS signal for a frame and the assertion of DSPI_PCS signal for the next frame. See Figure 31-15 for an illustration of the delay after transfer. DSPI_CTAR*n*[PDT, DT] fields select the delay after transfer by the formula:

$$t_{DT} = \frac{1}{f_{SYS/2}} \times PDT \times DT \qquad \textbf{\textit{Eqn. 31-4}}$$

Table 31-16 shows an example of the computed delay after transfer.

Table 31-16. Delay after Transfer Computation Example

PDT	Prescaler Value	DT	Scaler Value	$f_{SYS/2}$	Delay after Transfer
01	3	1110	32768	100 MHz	0.98 ms

31.4.3.5 Peripheral Chip Select Strobe Enable ($\overline{\text{PCSS}}$)

$\overline{\text{DSPI\_PCSS}}$ signal provides a delay to allow DSPI_PCS*n* signals to settle after transitioning, which avoids glitches. When DSPI is in master mode and DSPI_MCR[PCSSE] bit is set, $\overline{\text{DSPI\_PCSS}}$ provides a signal for an external demultiplexer to decode the DSPI_PCS[0:3] signals into as many as 16 glitch-free PCS signals. Figure 31-14 shows the timing of the $\overline{\text{DSPI\_PCSS}}$ signal relative to DSPI_PCS*n* signals.

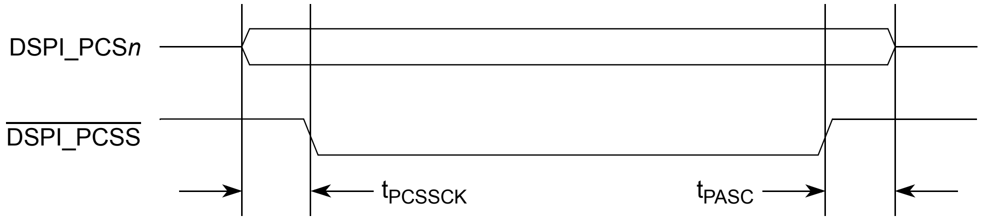


Figure 31-14. Peripheral Chip Select Strobe Timing

The delay between assertion of DSPI_PCS*n* signals and assertion of $\overline{\text{DSPI\_PCSS}}$ is selected by the DSPI_CTAR*n*[PCSSCK] field based on the following:

$$t_{PCSSCK} = \frac{1}{f_{SYS/2}} \times PCSSCK \qquad \textbf{\textit{Eqn. 31-5}}$$

At the end of transfer, delay between $\overline{\text{DSPI\_PCSS}}$ negation and DSPI_PCS*n* negation is selected by the DSPI_CTAR*n*[PASC] field based on the following:

$$t_{PASC} = \frac{1}{f_{SYS/2}} \times PASC \qquad \textbf{\textit{Eqn. 31-6}}$$

Table 31-17 shows an example of the computed t_{PCSSCK} delay.

Table 31-17. Peripheral Chip Select Strobe Assert Computation Example

PCSSCK	Prescaler	$f_{SYS/2}$	Delay before Transfer
0b11	7	100 MHz	70.0 ns

Table 31-18 shows an example of the computed t_{PASC} delay

Table 31-18. Peripheral Chip Select Strobe Negate Computation Example

PASC	Prescaler	$f_{SYS/2}$	Delay after Transfer
0b11	7	100 MHz	70.0 ns

NOTE

The $\overline{\text{DSPI\_PCSS}}$ signal is not supported when continuous DSPI_SCK is enabled (CONT = 1).

31.4.4 Transfer Formats

The serial communications clock (DSPI_SCK) signal and the DSPI_PCS*n* signals control the SPI serial communication. The DSPI_SCK signal provided by the master device synchronizes shifting and sampling of the data by the DSPI_SIN and DSPI_SOUT pins. The DSPI_PCS*n* signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the DSPI_CTAR*n*[CPOL, CPHA] bits select the polarity and phase of the DSPI_SCK signal. The polarity bit selects the idle state of the DSPI_SCK. The clock phase bit selects if the data on DSPI_SOUT is valid before or on the first DSPI_SCK edge.

When the DSPI is the bus slave, the DSPI_CTAR0[CPOL, CPHA] bits select the polarity and phase of the serial clock. Even though the bus slave does not control the DSPI_SCK signal, clock polarity, clock phase, and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

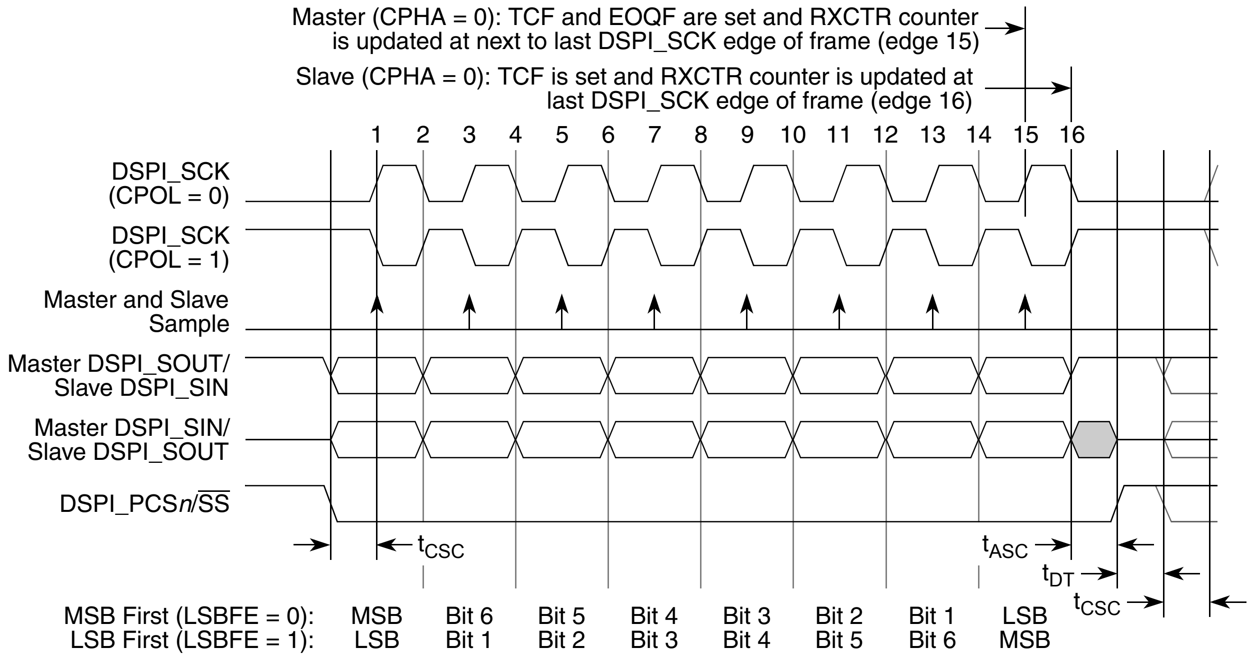
The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The DSPI_MCR[MTFE] bit selects between classic SPI format and modified transfer format. The classic SPI formats are described in Section 31.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” and Section 31.4.4.2, “Classic SPI Transfer Format (CPHA = 1).” The modified transfer formats are described in Section 31.4.4.3, “Modified SPI Transfer Format (MTFE = 1, CPHA = 0)” and Section 31.4.4.4, “Modified SPI Transfer Format (MTFE = 1, CPHA = 1).”

31.4.4.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in Figure 31-15 communicates with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their DSPI_SIN pins on the odd-numbered DSPI_SCK edges and change the data on their DSPI_SOUT pins on the even-numbered DSPI_SCK edges.


Master (CPHA = 0): TCF and EOQF are set and RXCTR counter is updated at next to last DSPI_SCK edge of frame (edge 15)
Slave (CPHA = 0): TCF is set and RXCTR counter is updated at last DSPI_SCK edge of frame (edge 16)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
DSPI_SCK (CPOL = 0)
DSPI_SCK (CPOL = 1)
Master and Slave Sample
Master DSPI_SOUT/ Slave DSPI_SIN
Master DSPI_SIN/ Slave DSPI_SOUT
DSPI_PCSn/SS
tCSC
tASC
tDT
tCSC
MSB First (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB First (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB

t_{CSC} = PCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS idle time).

Figure 31-15. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the DSPI_SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its DSPI_SOUT pin. After the t_{CSC} delay elapses, the master outputs the first edge of DSPI_SCK. The master and slave devices use this edge to sample the first input data bit on their serial data input signals. At the second edge of the DSPI_SCK, the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame, the master and the slave sample their DSPI_SIN pins on the odd-numbered clock edges and change the data on their DSPI_SOUT pins on the even-numbered clock edges. After the last clock edge occurs, a delay of t_{ASC} is inserted before the master negates the DSPI_PCS*n* signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

If DSPI_CTAR*n*[CPHA] is cleared:

- At the next to last serial clock edge of the frame (edge 15 of Figure 31-15)
  - Master’s TCF and EOQF are set and RXCTR counter is updated
- At the last serial clock edge of the frame (edge 16 of Figure 31-15)
  - Slave’s TCF is set and RXCTR counter is updated

31.4.4.2 Classic SPI Transfer Format (CPHA = 1)

The transfer format shown in Figure 31-16 communicates with peripheral SPI slave devices that require the first DSPI_SCK edge before the first data bit becomes available on the slave DSPI_SOUT pin. In this format, the master and slave devices change the data on their DSPI_SOUT pins on the odd-numbered DSPI_SCK edges and sample the data on their DSPI_SIN pins on the even-numbered DSPI_SCK edges.

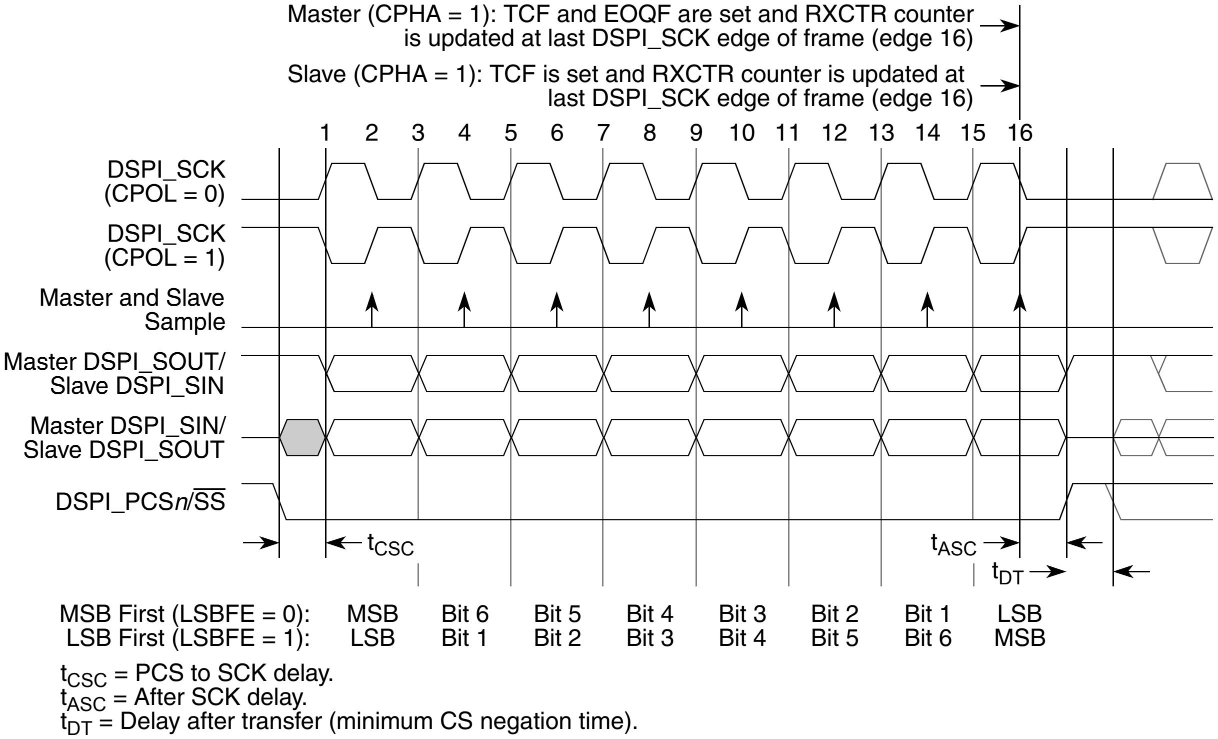

Master (CPHA = 1): TCF and EOQF are set and RXCTR counter is updated at last DSPI_SCK edge of frame (edge 16)
Slave (CPHA = 1): TCF is set and RXCTR counter is updated at last DSPI_SCK edge of frame (edge 16)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
DSPI_SCK (CPOL = 0)
DSPI_SCK (CPOL = 1)
Master and Slave Sample
Master DSPI_SOUT/ Slave DSPI_SIN
Master DSPI_SIN/ Slave DSPI_SOUT
DSPI_PCSn/SS
t_{CSC}
t_{ASC}
t_{DT}
MSB First (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB First (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB
t_{CSC} = PCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 31-16. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the DSPI_PCS*n* signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first DSPI_SCK edge and places valid data on the master DSPI_SOUT pin. The slave responds to the first DSPI_SCK edge by placing its first data bit on its slave DSPI_SOUT pin.

At the second edge of the DSPI_SCK, the master and slave sample their DSPI_SIN pins. For the rest of the frame, the master and the slave change the data on their DSPI_SOUT pins on the odd-numbered clock edges and sample their DSPI_SIN pins on the even-numbered clock edges. After the last clock edge occurs,1 a delay of t_{ASC} is inserted before the master negates the DSPI_PCS*n* signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

If DSPI_CTAR*n*[CPHA] is set:

- At the last serial clock edge (edge 16 of Figure 31-16)
  - — Master's EOQF and TCF are set
  - — Slave's TCF is set
  - — Master's and slave's RXCTR counters are updated

31.4.4.3 Modified SPI Transfer Format (MTFE = 1, CPHA = 0)

In this modified transfer format, the master and the slave sample later in the DSPI_SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the DSPI_SCK period as the DSPI_SCK period decreases with increasing baud rates.

NOTE

> For correct operation of the modified transfer format, thoroughly analyze the SPI link timing budget.

The master and the slave place data on the DSPI_SOUT pins at the assertion of the DSPI_PCS*n* signal. After the PCS to SCK delay has elapsed, the first DSPI_SCK edge is generated. The slave samples the master DSPI_SOUT signal on every odd numbered DSPI_SCK edge. The slave also places new data on the slave DSPI_SOUT on every odd numbered clock edge.

The master places its second data bit on the DSPI_SOUT line one system clock after odd numbered SDSPI_CK edge. Writing to the DSPI_MCR[SMPL_PT] field selects the point where the master samples the slave DSPI_SOUT. Table 31-19 lists the number of system clock cycles between the active edge of DSPI_SCK and the master sample point for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 31-19. Delayed Master Sample Point

SMPL_PT	Number of System Clock Cycles between Odd-numbered Edge of SCK and Sampling of SIN
00	0
01	1
10	2
11	Reserved

Figure 31-17 shows the modified transfer format for CPHA is cleared. Only the condition where CPOL is cleared is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

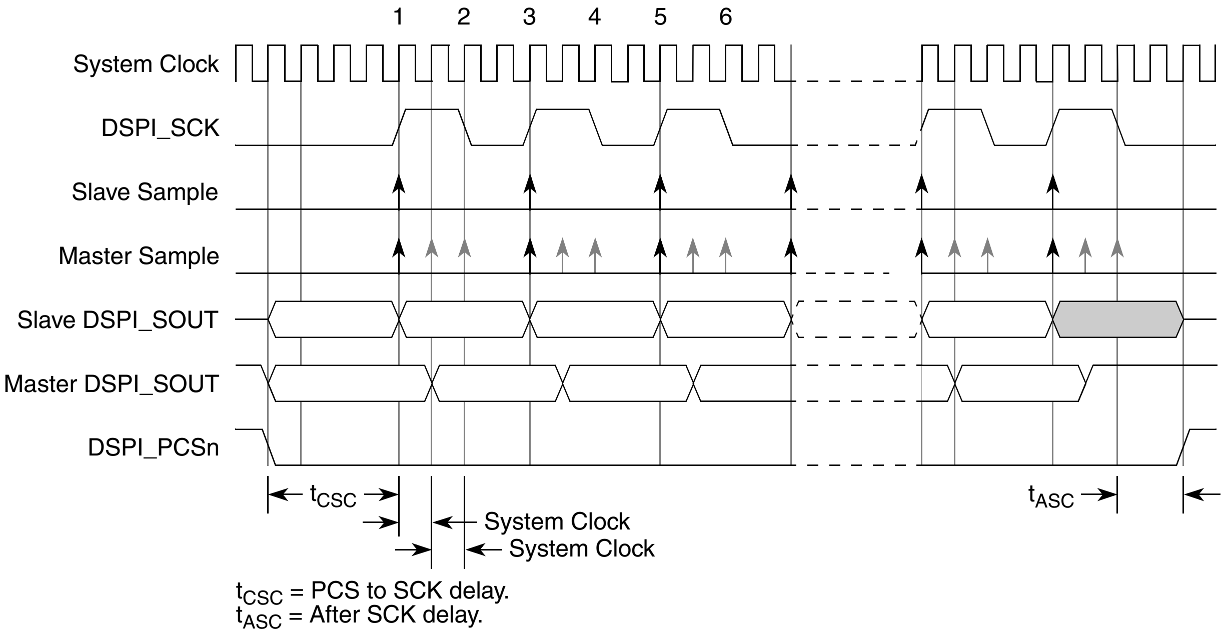


Figure 31-17. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, Fsck = Fsys/4)

31.4.4.4 Modified SPI Transfer Format (MTFE = 1, CPHA = 1)

Figure 31-18 shows the modified transfer format for CPHA is set. Only the condition where CPOL is cleared is described. At the start of a transfer, the DSPI asserts the DSPI_PCS*n* signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their DSPI_SOUT pins at the first edge of DSPI_SCK. The slave samples the master DSPI_SOUT signal on the even numbered edges of DSPI_SCK. The master samples the slave DSPI_SOUT signal on the odd numbered DSPI_SCK edges starting with the third DSPI_SCK edge. The slave samples the last bit on the last edge of the DSPI_SCK. The master samples the last slave DSPI_SOUT bit one half DSPI_SCK cycle after the last edge of DSPI_SCK. No clock edge is visible on the master DSPI_SCK pin during the sampling of the last bit. The SCK to PCS delay must be greater or equal to half of the DSPI_SCK period.

NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

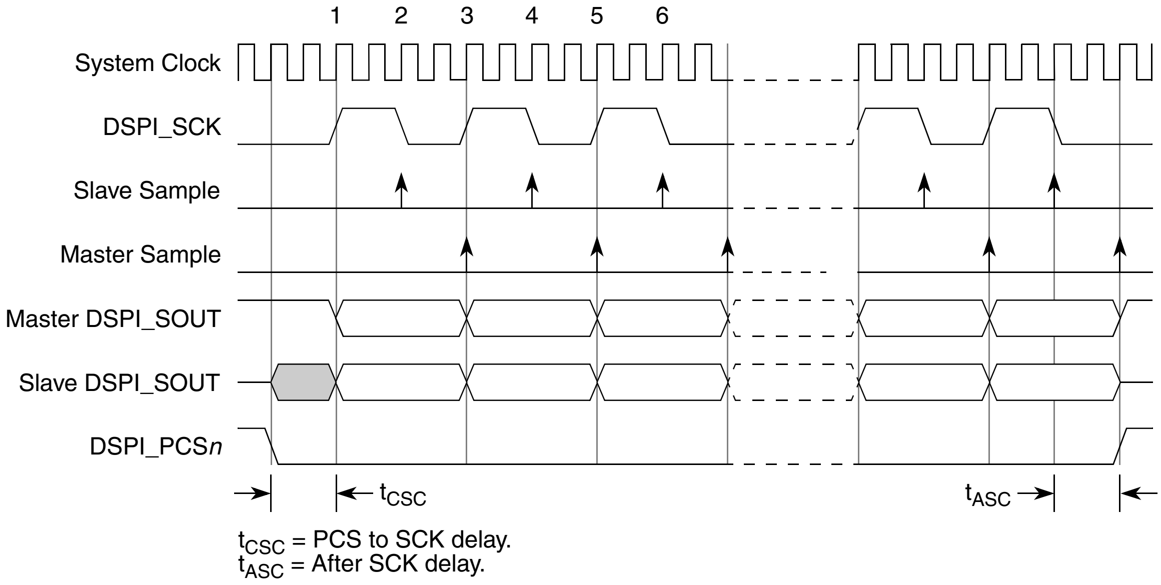


Figure 31-18. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, Fsck = Fsys/4)

31.4.4.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the DSPI_PUSHR[CONT] bit.

When CONT is cleared, DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the DSPI_MCR[PCSIS] field. Figure 31-19 shows the timing diagram for two four-bit transfers with CPHA set and CONT cleared.

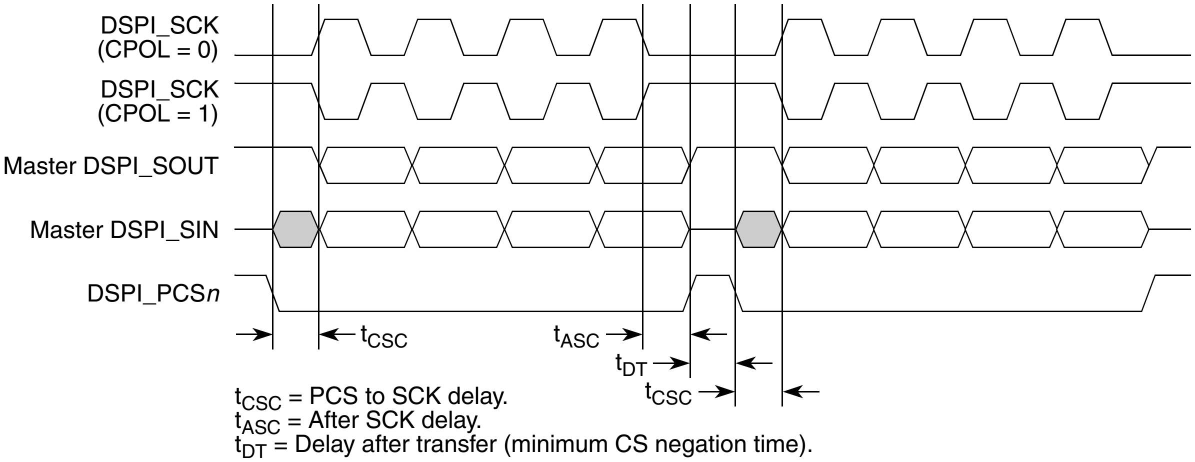

DSPI_SCK
(CPOL = 0)
DSPI_SCK
(CPOL = 1)
Master DSPI_SOUT
Master DSPI_SIN
DSPI_PCSn
t_{CSC}
t_{ASC}
t_{DT}
t_{CSC}
t_{CSC} = PCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 31-19. Example of Non-Continuous Format (CPHA=1, CONT=0)

When CONT is set and the DSPI_PCS*n* signal for the next transfer the same as for the current transfer, DSPI_PCS*n* signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers. Figure 31-20 shows the timing diagram for two four-bit transfers with CPHA and CONT set.

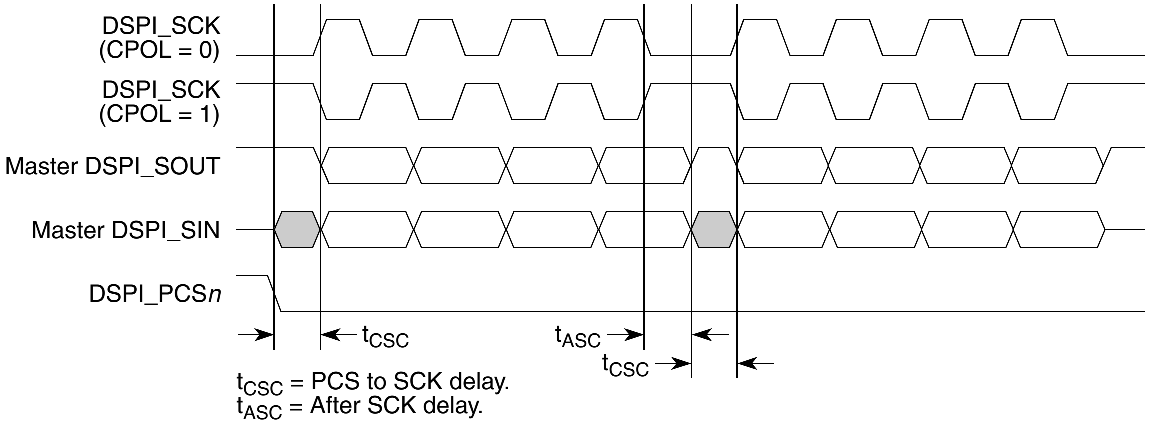


Figure 31-20. Example of Continuous Transfer (CPHA = 1, CONT = 1)

In Figure 31-20, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}. It does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching DSPI_CTAR*n* registers between frames while using continuous selection can cause errors in the transfer. The DSPI_PCS*n* signal must be negated before DSPI_CTAR is switched.

When CONT is set and the DSPI_PCS*n* signals for the next transfer are different from the present transfer, the DSPI_PCS*n* signals behave as if the CONT bit was cleared.

31.4.4.6 Clock Polarity Switching between DSPI Transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame. In Figure 31-21, time A shows the one clock interval. Time B is user programmable from a minimum of 2 system clocks. See Section 31.3.3, "DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTARn)."

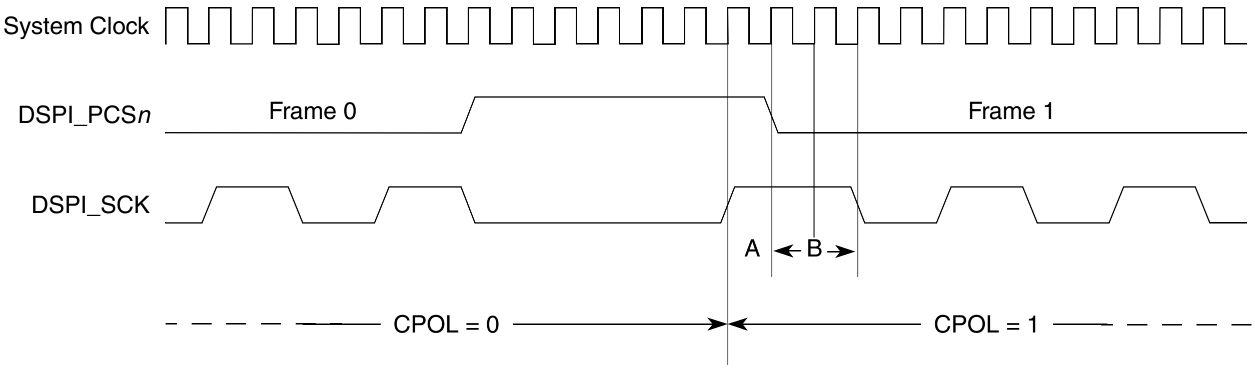


Figure 31-21. Polarity Switching between Frames

31.4.5 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous DSPI_SCK signal for slave peripherals that require a continuous clock. Continuous SCK is enabled by setting the DSPI_MCR[CONT_SCKE] bit.

Continuous SCK is only supported if CPHA is set. Clearing CPHA is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- DSPI_CTAR0 is used initially. At the start of each SPI frame transfer, the DSPI_CTAR*n* specified by the CTAS field for the frame is used.
- The currently selected DSPI_CTAR*n* remains in use until the start of a frame with a different DSPI_CTAR*n* specified, or the continuous SCK mode is terminated.

It is recommended that the baud rate is the same for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the PCS to SCK delay and the after SCK delay. The delay after transfer is fixed at one DSPI_SCK cycle. Figure 31-22 shows timing diagram for continuous SCK format with continuous selection disabled.

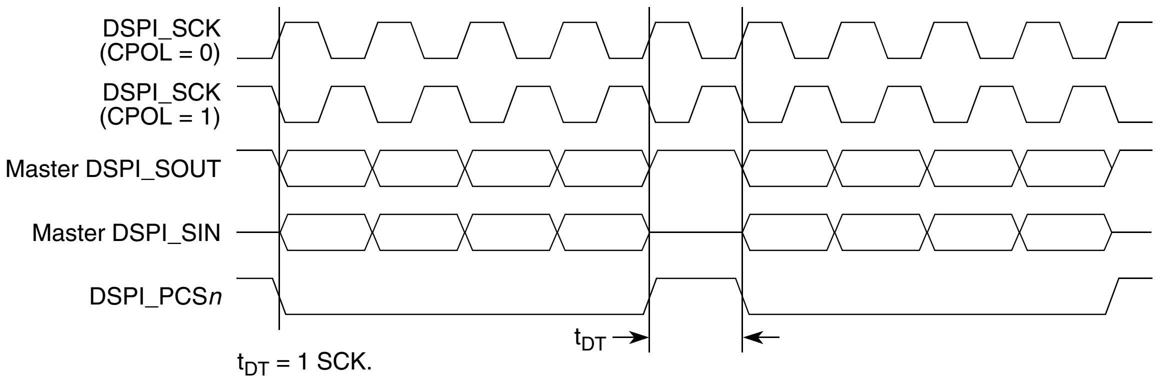


Figure 31-22. Continuous SCK Timing Diagram (CONT= 0)

If the CONT bit in the TX FIFO entry is set, DSPI_PCS*n* remains asserted between the transfers when the DSPI_PCS*n* signal for the next transfer is the same as for the current transfer. Figure 31-23 shows timing diagram for continuous SCK format with continuous selection enabled.

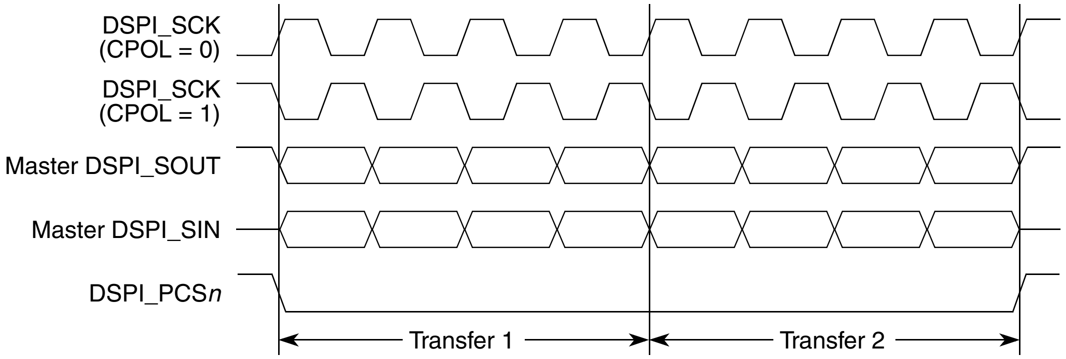


Figure 31-23. Continuous SCK Timing Diagram (CONT=1)

31.4.6 Interrupts/DMA Requests

The DSPI has six conditions that can only generate interrupt requests and two conditions that can generate an interrupt or DMA request. Table 31-20 lists these conditions.

Table 31-20. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	—
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	—
TX FIFO underflow has occurred	TFUF	X	—
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow has occurred	RFOF	X	—

Table 31-20. Interrupt and DMA Request Conditions (continued)

Condition	Flag	Interrupt	DMA
A FIFO overrun has occurred[1]	TFUF OR RFOF	X	—
General DSPI condition[2]	Any of the above	X	—

[1] The FIFO overrun condition is created by OR-ing the TFUF and RFOF flags together.

[2] OR'd condition of any of the six flags.

Each condition has a flag bit and a request enable bit. The flag bits are described in Section 31.3.4, "DSPI Status Register (DSPI_SR)," and the request enable bits are described in Section 31.3.5, "DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)." The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the DSPI_RSER[TFFF_DIRS, RFDF_DIRS] bits.

31.4.6.1 End of Queue Interrupt Request (EOQF)

The end of queue request indicates end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the DSPI_RSER[EOQF_RE] bit is set. See the EOQ bit description in Section 31.3.4, "DSPI Status Register (DSPI_SR)." Refer to Figure 31-15 and Figure 31-16 that illustrate when EOQF is set.

31.4.6.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the DSPI_RSER[TFFF_RE] bit is set. The DSPI_RSER[TFFF_DIRS] bit selects whether a DMA request or an interrupt request is generated.

31.4.6.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the DSPI_RSER[TCF_RE] bit is set. See the TCF bit description in Section 31.3.4, "DSPI Status Register (DSPI_SR)." Refer to Figure 31-15 and Figure 31-16 that illustrate when TCF is set.

31.4.6.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the DSPI_RSER[TFUF_RE] bit is set, an interrupt request is generated.

31.4.6.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the DSPI_RSER[RFDF_RE] bit is set. The DSPI_RSER[RFDF_DIRS] bit selects whether a DMA request or an interrupt request is generated.

31.4.6.6 Receive FIFO Overflow Interrupt Request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The DSPI_RSER[RFOF_RE] bit must be set for the interrupt request to be generated.

Depending on the state of the DSPI_MCR[ROOE] bit, data from the transfer that generated overflow is ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, incoming data is ignored.

31.4.6.7 FIFO Overrun Request (TFUF) or (RFOF)

The FIFO overrun request indicates at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing the RX FIFO overflow and TX FIFO underflow signals.

31.4.7 Power Saving Features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

31.4.7.1 Module Disable Mode

Module disable mode is a mode the DSPI can enter to save power. Host software can initiate the module disable mode by setting DSPI_MCR[MDIS]. The MDIS bit is set at reset.

In module disable mode, the DSPI is in a dormant state, but the memory-mapped registers remain accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any affect in module disable mode. Changes to the DSPI_MCR[DIS_TXF, DIS_RXF] fields do not have any affect in module disable mode. In module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no effect. Writing to the DSPI_TCR during module disable mode does not have any affect. Interrupt and DMA request signals cannot be cleared while in module disable mode.

31.4.7.2 Slave Interface Signal Gating

The DSPI's module enable signal gates slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

31.5 Initialization/Application Information

31.5.1 How to Change Queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag, DSPI_SR[EOQF] is set.
3. The setting of the EOQF flag disables serial transmission and serial reception of data, putting the DSPI in the stopped state. The TXRXS bit is cleared to indicate the stopped state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the DSPI_SR[RXCNT] bit or by checking the DSPI_SR[RFDF] bit after each read operation of the DSPI_POPR register.
7. Modify DMA descriptor of TX and RX channels for new queues.
8. Flush TX FIFO by writing a 1 to the DSPI_MCR[CLR_TXF] bit; Flush RX FIFO by writing a 1 to the DSPI_MCR[CLR_RXF] bit.
9. Clear transfer count by setting the CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to the DSPI_TCR[SPI_TCNT] field.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

31.5.2 Switching Master and Slave Mode

When changing modes in the DSPI, follow the steps below to guarantee proper operation.

1. Halt the DSPI by setting DSPI_MCR[HALT].
2. Clear the transmit and receive FIFOs by writing a 1 to the CLR_TXF and CLR_RXF bits in DSPI_MCR.
3. Set the appropriate mode in DSPI_MCR[MSTR] and enable the DSPI by clearing DSPI_MCR[HALT].

31.5.3 Baud Rate Settings

Table 31-21 shows the baud rate generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI_CTAR*n* registers. The values calculated assume a 100 MHz system frequency.

Table 31-21. Baud Rate Values

		Baud Rate Divider Prescaler Values (DSPI_CTAR*n*[PBR])			
		2	**3**	**5**	**7**
Baud Rate Scaler Values (DSPI_CTAR*n*[BR])	**2**	25.0MHz	16.7MHz	10.0MHz	7.14MHz
	4	12.5MHz	8.33MHz	5.00MHz	3.57MHz
	6	8.33MHz	5.56MHz	3.33MHz	2.38MHz
	8	6.25MHz	4.17MHz	2.50MHz	1.79MHz
	16	3.12MHz	2.08MHz	1.25MHz	893kHz
	32	1.56MHz	1.04MHz	625kHz	446kHz
	64	781kHz	521kHz	312kHz	223kHz
	128	391kHz	260kHz	156kHz	112kHz
	256	195kHz	130kHz	78.1kHz	55.8kHz
	512	97.7kHz	65.1kHz	39.1kHz	27.9kHz
	1024	48.8kHz	32.6kHz	19.5kHz	14.0kHz
	2048	24.4kHz	16.3kHz	9.77kHz	6.98kHz
	4096	12.2kHz	8.14kHz	4.88kHz	3.49kHz
	8192	6.10kHz	4.07kHz	2.44kHz	1.74kHz
	16384	3.05kHz	2.04kHz	1.22kHz	872Hz
	32768	1.53kHz	1.02kHz	610Hz	436Hz

31.5.4 Delay Settings

Table 31-22 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR*n* registers. The values calculated assume a 100 MHz system frequency.

Table 31-22. Delay Values

		Delay Prescaler Values (DSPI_CTAR*n*[PBR])			
		1	**3**	**5**	**7**
Delay Scaler Values (DSPI_CTAR*n*[DT])	**2**	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

31.5.5 Calculation of FIFO Pointer Addresses

Complete visibility of the TX and RX FIFO contents is available through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO, the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO, the first-in pointer is the pop next pointer (POPNXTPTR).

Figure 31-24 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See Section 31.4.2.4, “TX FIFO Buffering Mechanism,” and Section 31.4.2.5, “RX FIFO Buffering Mechanism,” for details on the FIFO operation.

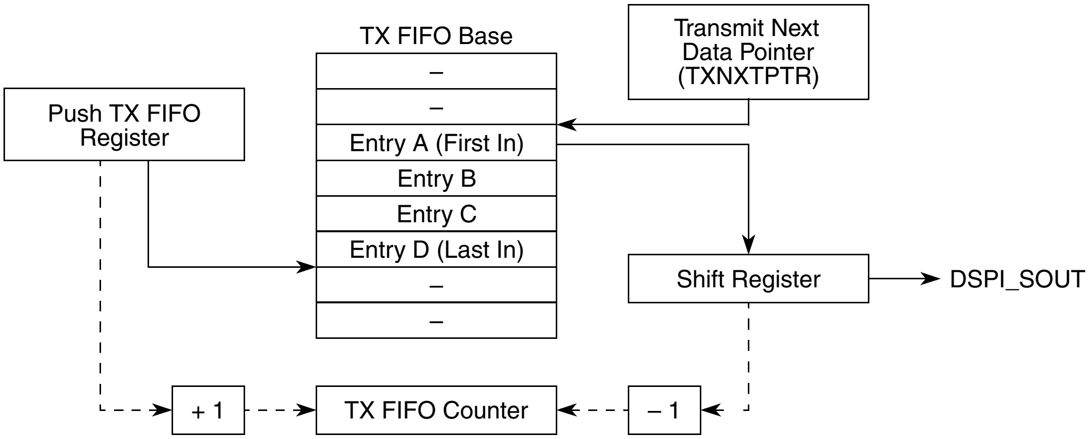


Figure 31-24. TX FIFO Pointers and Counter

31.5.5.1 Address Calculation for the First-in and Last-in Entries in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

First-in entry address = TXFIFO base + 4 × (TXNXTPTR)

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Last-in entry address = TX FIFO base + 4 × [(TXCTR + TXNXTPTR - 1) modulo TX FIFO depth]

where:

TX FIFO base: base address of TX FIFO

TXCTR: TX FIFO counter

TXNXTPTR: transmit next pointer

TX FIFO depth: 16

31.5.5.2 Address Calculation for the First-in and Last-in Entries in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

First-in entry address = RX FIFO base + 4 × (POPNXTPTR)

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Last-in entry address = RX FIFO base + 4 × [(RXCTR + POPNXTPTR - 1) modulo RX FIFO depth]

RX FIFO base: base address of RX FIFO

RXCTR: RX FIFO counter

POPNXTPTR: pop next pointer

RX FIFO depth: 16

Chapter 32
UART Modules

32.1 Introduction

This chapter describes the use of the three universal asynchronous receiver/transmitters (UARTs) and includes programming examples.

NOTE

> The designation *n* appears throughout this section to refer to registers or signals associated with one of the three identical UART modules: UART0, UART1, or UART2.

32.1.1 Overview

The internal bus clock can clock each of the three independent UARTs, eliminating the need for an external UART clock. As Figure 32-1 shows, each UART module interfaces directly to the CPU and consists of:

- Serial communication channel
- Programmable clock generation
- Interrupt control logic and DMA request logic
- Internal channel control logic

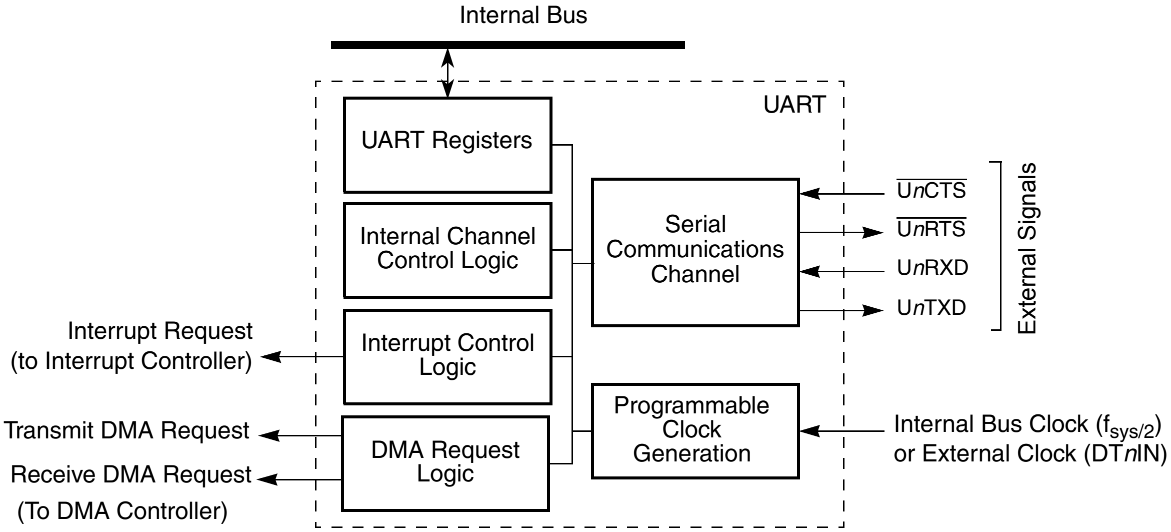

Internal Bus
UART
UART Registers
Internal Channel Control Logic
Interrupt Control Logic
DMA Request Logic
Serial Communications Channel
Programmable Clock Generation
UnCTS
UnRTS
UnRXD
UnTXD
External Signals
Interrupt Request (to Interrupt Controller)
Transmit DMA Request
Receive DMA Request
(To DMA Controller)
Internal Bus Clock ($f_{sys/2}$) or External Clock (DTnIN)

Figure 32-1. UART Block Diagram

NOTE

> The DT*n*IN pin can clock UART*n*. However, if the timers are operating and the UART uses DT*n*IN as a clock source, input capture mode is not available for that timer.

The serial communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter deriving an operating frequency from the internal bus clock or an external clock using the timer pin. The transmitter converts parallel data from the CPU to a serial bit stream, inserting appropriate start, stop, and parity bits. It outputs the resulting stream on the transmitter serial data output (U*n*TXD). See Section 32.4.2.1, "Transmitter."

The receiver converts serial data from the receiver serial data input (U*n*RXD) to parallel format, checks for a start, stop, and parity bits, or break conditions, and transfers the assembled character onto the bus during read operations. The receiver may be polled, interrupt driven, or use DMA requests for servicing. See Section 32.4.2.2, "Receiver."

NOTE

> The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, "Pin Multiplexing and Control") prior to configuring the UART module.

32.1.2 Features

The device contains three independent UART modules with:

- Each clocked by external clock or internal bus clock (eliminates need for an external UART clock)
- Full-duplex asynchronous/synchronous receiver/transmitter
- Quadruple-buffered receiver
- Double-buffered transmitter
- Independently programmable receiver and transmitter clock sources
- Programmable data format:
  - 5–8 data bits plus parity
  - Odd, even, no parity, or force parity
  - One, one-and-a-half, or two stop bits
- Each serial channel programmable to normal (full-duplex), automatic echo, local loopback, or remote loopback mode
- Automatic wake-up mode for multidrop applications
- Four maskable interrupt conditions
- All three UARTs have DMA request capability
- Parity, framing, and overrun error detection
- False-start bit detection
- Line-break detection and generation
- Detection of breaks originating in the middle of a character

- Start/end break interrupt/status

32.2 External Signal Description

Table 32-1 briefly describes the UART module signals.

Table 32-1. UART Module External Signals

Signal	Description
U*n*TXD	Transmitter Serial Data Output. U*n*TXD is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on U*n*TXD on the falling edge of the clock source, with the least significant bit (lsb) sent first.
U*n*RXD	Receiver Serial Data Input. Data received on U*n*RXD is sampled on the rising edge of the clock source, with the lsb received first.
$\overline{\mathrm{U}\mathit{n}\mathrm{CTS}}$	Clear-to- Send. This input can generate an interrupt on a change of state.
$\overline{\mathrm{U}\mathit{n}\mathrm{RTS}}$	Request-to-Send. This output can be programmed to be negated or asserted automatically by the receiver or the transmitter. When connected to a transmitter's $\overline{\mathrm{U}\mathit{n}\mathrm{CTS}}$, $\overline{\mathrm{U}\mathit{n}\mathrm{RTS}}$ can control serial data flow.

Figure 32-2 shows a signal configuration for a UART/RS-232 interface.

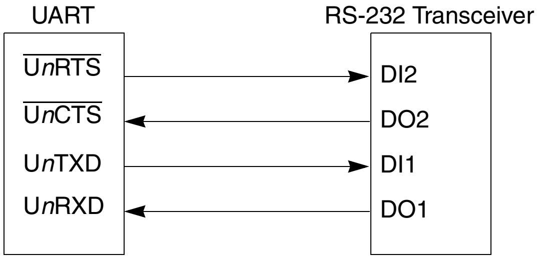

UART
RS-232 Transceiver
$\overline{\mathrm{U}\mathit{n}\mathrm{RTS}}$
$\overline{\mathrm{U}\mathit{n}\mathrm{CTS}}$
U*n*TXD
U*n*RXD
DI2
DO2
DI1
DO1

Figure 32-2. UART/RS-232 Interface

32.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in Section 32.5, "Initialization/Application Information," describe basic UART module programming. Writing control bytes into the appropriate registers controls the operation of the UART module.

NOTE

UART registers are accessible only as bytes.

NOTE

Interrupt can mean an interrupt request asserted to the CPU or a DMA request.

Table 32-2. UART Module Memory Map

Address	Register	Width (bit)	Access	Reset Value	Section/Page
UART0 UART1 UART2					
0xFC06_0000 0xFC06_4000 0xFC06_8000	UART Mode Registers[1] (UMR1*n*), (UMR2*n*)	8	R/W	0x00	32.3.1/32-5 32.3.2/32-6
0xFC06_0004 0xFC06_4004 0xFC06_8004	UART Status Register (USR*n*)	8	R	0x00	32.3.3/32-8
	UART Clock Select Register[1](UCSR*n*)	8	W	See Section	32.3.4/32-9
0xFC06_0008 0xFC06_4008 0xFC06_8008	UART Command Registers (UCR*n*)	8	W	0x00	32.3.5/32-9
0xFC06_000C 0xFC06_400C 0xFC06_800C	UART Receive Buffers (URB*n*)	8	R	0xFF	32.3.6/32-11
	UART Transmit Buffers (UTB*n*)	8	W	0x00	32.3.7/32-12
0xFC06_0010 0xFC06_4010 0xFC06_8010	UART Input Port Change Register (UIPCR*n*)	8	R	See Section	32.3.8/32-12
	UART Auxiliary Control Register (UACR*n*)	8	W	0x00	32.3.9/32-13
0xFC06_0014 0xFC06_4014 0xFC06_8014	UART Interrupt Status Register (UISR*n*)	8	R	0x00	32.3.10/32-13
	UART Interrupt Mask Register (UIMR*n*)	8	W	0x00	
0xFC06_0018 0xFC06_4018 0xFC06_8018	UART Baud Rate Generator Register (UBG1*n*)	8	W[2]	0x00	32.3.11/32-15
0xFC06_001C 0xFC06_401C 0xFC06_801C	UART Baud Rate Generator Register (UBG2*n*)	8	W[2]	0x00	32.3.11/32-15
0xFC06_0034 0xFC06_4034 0xFC06_8034	UART Input Port Register (UIP*n*)	8	R	0xFF	32.3.12/32-15
0xFC06_0038 0xFC06_4038 0xFC06_8038	UART Output Port Bit Set Command Register (UOP1*n*)	8	W[2]	0x00	32.3.13/32-16
0xFC06_003C 0xFC06_403C 0xFC06_803C	UART Output Port Bit Reset Command Register (UOP0*n*)	8	W[2]	0x00	32.3.13/32-16

[1] UMR1*n*, UMR2*n*, and UCSR*n* must be changed only after the receiver/transmitter is issued a software reset command. If operation is not disabled, undesirable results may occur.

[2] Reading this register results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

32.3.1 UART Mode Registers 1 (UMR1*n*)

The UMR1*n* registers control UART module configuration. UMR1*n* can be read or written when the mode register pointer points to it, at RESET or after a RESET MODE REGISTER POINTER command using UCR*n*[MISC]. After UMR1*n* is read or written, the pointer points to UMR2*n*.

Address: 0xFC06_0000 (UMR10)
0xFC06_4000 (UMR11)
0xFC06_8000 (UMR12)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	RXRTS	RXIRQ/ FFULL	ERR	PM		PT	B/C	
Reset:	0	0	0	0	0	0	0	0

[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 32-3. UART Mode Registers 1 (UMR1*n*)

Table 32-3. UMR1*n* Field Descriptions

Field	Description
7 RXRTS	Receiver request-to-send. Allows the $\overline{\mathrm{U}n\mathrm{RTS}}$ output to control the $\overline{\mathrm{U}n\mathrm{CTS}}$ input of the transmitting device to prevent receiver overrun. If the receiver and transmitter are incorrectly programmed for $\overline{\mathrm{U}n\mathrm{RTS}}$ control, $\overline{\mathrm{U}n\mathrm{RTS}}$ control is disabled for both. Transmitter RTS control is configured in UMR2*n*[TXRTS]. 0 The receiver has no effect on $\overline{\mathrm{U}n\mathrm{RTS}}$. 1 When a valid start bit is received, $\overline{\mathrm{U}n\mathrm{RTS}}$ is negated if the UART's FIFO is full. $\overline{\mathrm{U}n\mathrm{RTS}}$ is reasserted when the FIFO has an empty position available.
6 RXIRQ/ FFULL	Receiver interrupt select. 0 RXRDY is the source generating interrupt or DMA requests. 1 FFULL is the source generating interrupt or DMA requests.
5 ERR	Error mode. Configures the FIFO status bits, USR*n*[RB,FE,PE]. 0 Character mode. The USR*n* values reflect the status of the character at the top of the FIFO. ERR must be 0 for correct A/D flag information when in multidrop mode. 1 Block mode. The USR*n* values are the logical OR of the status for all characters reaching the top of the FIFO since the last RESET ERROR STATUS command for the UART was issued. See Section 32.3.5, "UART Command Registers (UCRn)."
4–3 PM	Parity mode. Selects the parity or multidrop mode for the UART. The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. The value of PM affects PT, as shown below.

Table 32-3. UMR1*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
PT</td><td>Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11).
<table>
<tr><th>PM</th><th>Parity Mode</th><th>Parity Type (PT= 0)</th><th>Parity Type (PT= 1)</th></tr>
<tr><td>00</td><td>With parity</td><td>Even parity</td><td>Odd parity</td></tr>
<tr><td>01</td><td>Force parity</td><td>Low parity</td><td>High parity</td></tr>
<tr><td>10</td><td>No parity</td><td colspan="2">N/A</td></tr>
<tr><td>11</td><td>Multidrop mode</td><td>Data character</td><td>Address character</td></tr>
</table>
</td></tr>
<tr><td>1–0
B/C</td><td>Bits per character. Selects the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits.
00 5 bits
01 6 bits
10 7 bits
11 8 bits</td></tr>
</table>

32.3.2 UART Mode Register 2 (UMR2*n*)

The UMR2*n* registers control UART module configuration. UMR2*n* can be read or written when the mode register pointer points to it, which occurs after any access to UMR1*n*. UMR2*n* accesses do not update the pointer.

Address: 0xFC06_0000 (UMR20)
0xFC06_4000 (UMR21)
0xFC06_8000 (UMR22)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	CM		TXRTS	TXCTS	SB			
Reset:	0	0	0	0	0	0	0	0

[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 32-4. UART Mode Registers 2 (UMR2*n*)

Table 32-4. UMR2*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–6
CM</td><td>Channel mode. Selects a channel mode. Section 32.4.3, “Looping Modes,” describes individual modes.
00 Normal
01 Automatic echo
10 Local loopback
11 Remote loopback</td></tr>
<tr><td>5
TXRTS</td><td>Transmitter ready-to-send. Controls negation of $\overline{\text{U}n\text{RTS}}$ to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same UART for $\overline{\text{U}n\text{RTS}}$ control is not permitted and disables $\overline{\text{U}n\text{RTS}}$ control for both.
0 The transmitter has no effect on $\overline{\text{U}n\text{RTS}}$.
1 In applications where the transmitter is disabled after transmission completes, setting this bit automatically clears UOP[RTS] one bit time after any characters in the transmitter shift and holding registers are completely sent, including the programmed number of stop bits.</td></tr>
<tr><td>4
TXCTS</td><td>Transmitter clear-to-send. If TXCTS and TXRTS are set, TXCTS controls the operation of the transmitter.
0 $\overline{\text{U}n\text{CTS}}$ has no effect on the transmitter.
1 Enables clear-to-send operation. The transmitter checks the state of $\overline{\text{U}n\text{CTS}}$ each time it is ready to send a character. If $\overline{\text{U}n\text{CTS}}$ is asserted, the character is sent; if it is deasserted, the signal U<i>n</i>TXD remains in the high state and transmission is delayed until $\overline{\text{U}n\text{CTS}}$ is asserted. Changes in $\overline{\text{U}n\text{CTS}}$ as a character is being sent do not affect its transmission.</td></tr>
<tr><td>3–0
SB</td><td>Stop-bit length control. Selects length of stop bit appended to the transmitted character. Stop-bit lengths of 9/16 to 2 bits are programmable for 6–8 bit characters. Lengths of 1-1/16 to 2 bits are programmable for 5-bit characters. In all cases, the receiver checks only for a high condition at the center of the first stop-bit position, one bit time after the last data bit or after the parity bit, if parity is enabled. If an external 1x clock is used for the transmitter, clearing bit 3 selects one stop bit and setting bit 3 selects two stop bits for transmission.
<table>
<tr><th>SB</th><th>5 Bits</th><th>6–8 Bits</th><th>SB</th><th>5–8 Bits</th></tr>
<tr><td>0000</td><td>1.063</td><td>0.563</td><td>1000</td><td>1.563</td></tr>
<tr><td>0001</td><td>1.125</td><td>0.625</td><td>1001</td><td>1.625</td></tr>
<tr><td>0010</td><td>1.188</td><td>0.688</td><td>1010</td><td>1.688</td></tr>
<tr><td>0011</td><td>1.250</td><td>0.750</td><td>1011</td><td>1.750</td></tr>
<tr><td>0100</td><td>1.313</td><td>0.813</td><td>1100</td><td>1.813</td></tr>
<tr><td>0101</td><td>1.375</td><td>0.875</td><td>1101</td><td>1.875</td></tr>
<tr><td>0110</td><td>1.438</td><td>0.938</td><td>1110</td><td>1.938</td></tr>
<tr><td>0111</td><td>1.500</td><td>1.000</td><td>1111</td><td>2.000</td></tr>
</table></td></tr>
</table>

32.3.3 UART Status Registers (USR*n*)

The USR*n* registers show the status of the transmitter, the receiver, and the FIFO.

Address: 0xFC06_0004 (USR0)
0xFC06_4004 (USR1)
0xFC06_8004 (USR2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	RB	FE	PE	OE	TXEMP	TXRDY	FFULL	RXRDY
W								
Reset:	0	0	0	0	0	0	0	0

Figure 32-5. UART Status Registers (USR*n*)

Table 32-5. USR*n* Field Descriptions

Field	Description
7 RB	Received break. The received break circuit detects breaks originating in the middle of a received character. However, a break in the middle of a character must persist until the end of the next detected character time. 0 No break was received. 1 An all-zero character of the programmed length was received without a stop bit. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until U*n*RXD returns to the high state for at least one-half bit time, which equals two successive edges of the UART clock. RB is valid only when RXRDY is set.
6 FE	Framing error. 0 No framing error occurred. 1 No stop bit was detected when the corresponding data character in the FIFO was received. The stop-bit check occurs in the middle of the first stop-bit position. FE is valid only when RXRDY is set.
5 PE	Parity error. Valid only if RXRDY is set. 0 No parity error occurred. 1 If UMR1*n*[PM] equals 0*x* (with parity or force parity), the corresponding character in the FIFO was received with incorrect parity. If UMR1*n*[PM] equals 11 (multidrop), PE stores the received address or data (A/D) bit. PE is valid only when RXRDY is set.
4 OE	Overrun error. Indicates whether an overrun occurs. 0 No overrun occurred. 1 One or more characters in the received data stream have been lost. OE is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, framing error status, and parity error, if any, are lost. The RESET ERROR STATUS command in UCR*n* clears OE.
3 TXEMP	Transmitter empty. 0 The transmit buffer is not empty. A character is shifted out, or the transmitter is disabled. The transmitter is enabled/disabled by programming UCR*n*[TC]. 1 The transmitter has underrun (the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission.
2 TXRDY	Transmitter ready. 0 The CPU loaded the transmitter holding register, or the transmitter is disabled. 1 The transmitter holding register is empty and ready for a character. TXRDY is set when a character is sent to the transmitter shift register or when the transmitter is first enabled. If the transmitter is disabled, characters loaded into the transmitter holding register are not sent.

Table 32-5. USR*n* Field Descriptions (continued)

Field	Description
1 FFULL	FIFO full. 0 The FIFO is not full but may hold up to two unread characters. 1 A character was received and the receiver FIFO is now full. Any characters received when the FIFO is full are lost.
0 RXRDY	Receiver ready. 0 The CPU has read the receive buffer and no characters remain in the FIFO after this read. 1 One or more characters were received and are waiting in the receive buffer FIFO.

32.3.4 UART Clock Select Registers (UCSR*n*)

The UCSRs select an external clock on the DTIN input (divided by 1 or 16) or a prescaled internal bus clock as the clocking source for the transmitter and receiver. See Section 32.4.1, “Transmitter/Receiver Clock Source.” The transmitter and receiver can use different clock sources. To use the internal bus clock for both, set UCSR*n* to 0xDD.

Address: 0xFC06_0004 (UCSR0)
0xFC06_4004 (UCSR1)
0xFC06_8004 (UCSR2)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	RCS				TCS			
Reset:	See Note				See Note			

Note: The RCS and TCS reset values are set so the receiver and transmitter use the prescaled internal bus clock as their clock source.

Figure 32-6. UART Clock Select Registers (UCSR*n*)

Table 32-6. UCSR*n* Field Descriptions

Field	Description
7–4 RCS	Receiver clock select. Selects the clock source for the receiver. 1101 Prescaled internal bus clock ($f_{sys/2}$) 1110 DT*n*IN divided by 16 1111 DT*n*IN
3–0 TCS	Transmitter clock select. Selects the clock source for the transmitter. 1101 Prescaled internal bus clock ($f_{sys/2}$) 1110 DT*n*IN divided by 16 1111 DT*n*IN

32.3.5 UART Command Registers (UCR*n*)

The UCRs supply commands to the UART. Only multiple commands that do not conflict can be specified in a single write to a UCR*n*. For example, RESET TRANSMITTER and ENABLE TRANSMITTER cannot be specified in one command.

Address: 0xFC06_0008 (UCR0)
0xFC06_4008 (UCR1)
0xFC06_8008 (UCR2)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	0	MISC			TC		RC	
Reset:	0	0	0	0	0	0	0	0

Figure 32-7. UART Command Registers (UCR*n*)

Table 32-7 describes UCR*n* fields and commands. Examples in Section 32.4.2, "Transmitter and Receiver Operating Modes," show how these commands are used.

Table 32-7. UCR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be cleared.</td></tr>
<tr><td>6–4
MISC</td><td>MISC Field (this field selects a single command)
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>000</td><td>NO COMMAND</td><td>—</td></tr>
<tr><td>001</td><td>RESET MODE REGISTER POINTER</td><td>Causes the mode register pointer to point to UMR1<i>n</i>.</td></tr>
<tr><td>010</td><td>RESET RECEIVER</td><td>Immediately disables the receiver, clears USR<i>n</i>[FFULL,RXRDY], and reinitializes the receiver FIFO pointer. No other registers are altered. Because it places the receiver in a known state, use this command instead of RECEIVER DISABLE when reconfiguring the receiver.</td></tr>
<tr><td>011</td><td>RESET TRANSMITTER</td><td>Immediately disables the transmitter and clears USR<i>n</i>[TXEMP,TXRDY]. No other registers are altered. Because it places the transmitter in a known state, use this command instead of TRANSMITTER DISABLE when reconfiguring the transmitter.</td></tr>
<tr><td>100</td><td>RESET ERROR STATUS</td><td>Clears USR<i>n</i>[RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received.</td></tr>
<tr><td>101</td><td>RESET BREAK – CHANGE INTERRUPT</td><td>Clears the delta break bit, UISR<i>n</i>[DB].</td></tr>
<tr><td>110</td><td>START BREAK</td><td>Forces U<i>n</i>TXD low. If the transmitter is empty, break may be delayed up to one bit time. If the transmitter is active, break starts when character transmission completes. Break is delayed until any character in the transmitter shift register is sent. Any character in the transmitter holding register is sent after the break. Transmitter must be enabled for the command to be accepted. This command ignores the state of $\overline{\mathrm{U}n\mathrm{CTS}}$.</td></tr>
<tr><td>111</td><td>STOP BREAK</td><td>Causes U<i>n</i>TXD to go high (mark) within two bit times. Any characters in the transmit buffer are sent.</td></tr>
</table>
</td></tr>
</table>

Table 32-7. UCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–2
TC</td><td>Transmit command field. Selects a single transmit command.
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>00</td><td>NO ACTION TAKEN</td><td>Causes the transmitter to stay in its current mode: if the transmitter is enabled, it remains enabled; if the transmitter is disabled, it remains disabled.</td></tr>
<tr><td>01</td><td>TRANSMITTER ENABLE</td><td>Enables operation of the UART's transmitter. USR<i>n</i>[TXEMP,TXRDY] are set. If the transmitter is already enabled, this command has no effect.</td></tr>
<tr><td>10</td><td>TRANSMITTER DISABLE</td><td>Terminates transmitter operation and clears USR<i>n</i>[TXEMP,TXRDY]. If a character is being sent when the transmitter is disabled, transmission completes before the transmitter becomes inactive. If the transmitter is already disabled, the command has no effect.</td></tr>
<tr><td>11</td><td>—</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
<tr><td>1–0
RC</td><td>Receive command field. Selects a single receive command.
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>00</td><td>NO ACTION TAKEN</td><td>Causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.</td></tr>
<tr><td>01</td><td>RECEIVER ENABLE</td><td>If the UART module is not in multidrop mode (UMR1<i>n</i>[PM] ≠ 11), RECEIVER ENABLE enables the UART's receiver and forces it into search-for-start-bit state. If the receiver is already enabled, this command has no effect.</td></tr>
<tr><td>10</td><td>RECEIVER DISABLE</td><td>Disables the receiver immediately. Any character being received is lost. The command does not affect receiver status bits or other control registers. If the UART module is programmed for local loopback or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, the command has no effect.</td></tr>
<tr><td>11</td><td>—</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
</table>

32.3.6 UART Receive Buffers (URB*n*)

The receive buffers contain one serial shift register and three receiver holding registers, which act as a FIFO. U*n*RXD is connected to the serial shift register. The CPU reads from the top of the FIFO while the receiver shifts and updates from the bottom when the shift register is full (see Figure 32-18). RB contains the character in the receiver.

Address: 0xFC06_000C (URB0)
0xFC06_400C (URB1)
0xFC06_800C (URB2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	RB							
W								
Reset:	1	1	1	1	1	1	1	1

Figure 32-8. UART Receive Buffer (URB*n*)

32.3.7 UART Transmit Buffers (UTB*n*)

The transmit buffers consist of the transmitter holding register and the transmitter shift register. The holding register accepts characters from the bus master if UART's USR*n*[TXRDY] is set. A write to the transmit buffer clears USR*n*[TXRDY], inhibiting any more characters until the shift register can accept more data. When the shift register is empty, it checks if the holding register has a valid character to be sent (TXRDY = 0). If there is a valid character, the shift register loads it and sets USR*n*[TXRDY] again. Writes to the transmit buffer when the UART's TXRDY is cleared and the transmitter is disabled have no effect on the transmit buffer.

Figure 32-9 shows UTB*n*. TB contains the character in the transmit buffer.

Address: 0xFC06_000C (UTB0)
0xFC06_400C (UTB1)
0xFC06_800C (UTB2)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	TB							
Reset:	0	0	0	0	0	0	0	0

Figure 32-9. UART Transmit Buffer (UTB*n*)

32.3.8 UART Input Port Change Registers (UIPCR*n*)

The UIPCRs hold the current state and the change-of-state for $\overline{\mathrm{U}n\mathrm{CTS}}$.

Address: 0xFC06_0010 (UIPCR0)
0xFC06_4010 (UIPCR1)
0xFC06_8010 (UIPCR2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	0	0	0	COS	1	1	1	CTS
W								
Reset:	0	0	0	0	1	1	1	$\overline{\mathrm{U}n\mathrm{CTS}}$

Figure 32-10. UART Input Port Changed Registers (UIPCR*n*)

Table 32-8. UIPCR*n* Field Descriptions

Field	Description
7–5	Reserved
4 COS	Change of state (high-to-low or low-to-high transition). 0 No change-of-state since the CPU last read UIPCR*n*. Reading UIPCR*n* clears UISR*n*[COS]. 1 A change-of-state longer than 25–50 µs occurred on the $\overline{\mathrm{U}n\mathrm{CTS}}$ input. UACR*n* can be programmed to generate an interrupt to the CPU when a change of state is detected.
3–1	Reserved
0 CTS	Current state of clear-to-send. Starting two serial clock periods after reset, CTS reflects the state of $\overline{\mathrm{U}n\mathrm{CTS}}$. If $\overline{\mathrm{U}n\mathrm{CTS}}$ is detected asserted at that time, COS is set, which initiates an interrupt if UACR*n*[IEC] is enabled. 0 The current state of the $\overline{\mathrm{U}n\mathrm{CTS}}$ input is asserted. 1 The current state of the $\overline{\mathrm{U}n\mathrm{CTS}}$ input is deasserted.

32.3.9 UART Auxiliary Control Register (UACR*n*)

The UACRs control the input enable.

Address: 0xFC06_0010 (UACR0)
0xFC06_4010 (UACR1)
0xFC06_8010 (UACR2)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	IEC
Reset:	0	0	0	0	0	0	0	0

Figure 32-11. UART Auxiliary Control Registers (UACR*n*)

Table 32-9. UACR*n* Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 IEC	Input enable control. 0 Setting the corresponding UIPCR*n* bit has no effect on UISR*n*[COS]. 1 UISR*n*[COS] is set and an interrupt is generated when the UIPCR*n*[COS] is set by an external transition on the $\overline{\mathrm{U}n\mathrm{CTS}}$ input (if UIMR*n*[COS] = 1).

32.3.10 UART Interrupt Status/Mask Registers (UISR*n*/UIMR*n*)

The UISRs provide status for all potential interrupt sources. UISR*n* contents are masked by UIMR*n*. If corresponding UISR*n* and UIMR*n* bits are set, internal interrupt output is asserted. If a UIMR*n* bit is cleared, state of the corresponding UISR*n* bit has no effect on the output.

The UISR*n* and UIMR*n* registers share the same space in memory. Reading this register provides the user with interrupt status, while writing controls the mask bits.

NOTE

True status is provided in the UISR*n* regardless of UIMR*n* settings. UISR*n* is cleared when the UART module is reset.

Address: 0xFC06_0014 (UISR0)
0xFC06_4014 (UISR1)
0xFC06_8014 (UISR2)

Access: User read/write

	7	6	5	4	3	2	1	0
R (UISR*n*)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
W (UIMR*n*)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
Reset:	0	0	0	0	0	0	0	0

Figure 32-12. UART Interrupt Status/Mask Registers (UISR*n*/UIMR*n*)

Table 32-10. UISR*n*/UIMR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7
COS</td><td>Change-of-state.
0 UIPCR<i>n</i>[COS] is not selected.
1 Change-of-state occurred on $\overline{\mathrm{U}n\mathrm{CTS}}$ and was programmed in UACR<i>n</i>[IEC] to cause an interrupt.</td></tr>
<tr><td>6–3</td><td>Reserved, must be cleared.</td></tr>
<tr><td>2
DB</td><td>Delta break.
0 No new break-change condition to report. Section 32.3.5, “UART Command Registers (UCRn),” describes the RESET BREAK-CHANGE INTERRUPT command.
1 The receiver detected the beginning or end of a received break.</td></tr>
<tr><td>1
FFULL/
RXRDY</td><td>Status of FIFO or receiver, depending on UMR1[FFULL/RXRDY] bit. Duplicate of USR<i>n</i>[FIFO] and USR<i>n</i>[RXRDY]
<table>
<tr><th rowspan="2">UIMR<i>n</i>
[FFULL/RXRDY]</th><th rowspan="2">UISR<i>n</i>
[FFULL/RXRDY]</th><th colspan="2">UMR1<i>n</i>[FFULL/RXRDY]</th></tr>
<tr><th>0 (RXRDY)</th><th>1 (FIFO)</th></tr>
<tr><td>0</td><td>0</td><td>Receiver not ready</td><td>FIFO not full</td></tr>
<tr><td>1</td><td>0</td><td>Receiver not ready</td><td>FIFO not full</td></tr>
<tr><td>0</td><td>1</td><td>Receiver is ready,
Do not interrupt</td><td>FIFO is full,
Do not interrupt</td></tr>
<tr><td>1</td><td>1</td><td>Receiver is ready,
interrupt</td><td>FIFO is full,
interrupt</td></tr>
</table>
</td></tr>
<tr><td>0
TXRDY</td><td>Transmitter ready. This bit is the duplication of USR<i>n</i>[TXRDY].
0 The transmitter holding register was loaded by the CPU or the transmitter is disabled. Characters loaded into the transmitter holding register when TXRDY is cleared are not sent.
1 The transmitter holding register is empty and ready to be loaded with a character.</td></tr>
</table>

32.3.11 UART Baud Rate Generator Registers (UBG1*n*/UBG2*n*)

The UBG1*n* registers hold the MSB, and the UBG2*n* registers hold the LSB of the preload value. UBG1*n* and UBG2*n* concatenate to provide a divider to the internal bus clock for transmitter/receiver operation, as described in Section 32.4.1.2.1, “Internal Bus Clock Baud Rates.”

Address: 0xFC06_0018 (UBG10)
0xFC06_4018 (UBG11)
0xFC06_8018 (UBG12)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	Divider MSB							
Reset:	0	0	0	0	0	0	0	0

Figure 32-13. UART Baud Rate Generator Registers (UBG1*n*)

Address: 0xFC06_001C (UBG20)
0xFC06_401C (UBG21)
0xFC06_801C (UBG22)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	Divider LSB							
Reset:	0	0	0	0	0	0	0	0

Figure 32-14. UART Baud Rate Generator Registers (UBG2*n*)

NOTE

The minimum value loaded on the concatenation of UBG1*n* with UBG2*n* is 0x0002. The UBG2*n* reset value of 0x00 is invalid and must be written to before the UART transmitter or receiver are enabled. UBG1*n* and UBG2*n* are write-only and cannot be read by the CPU.

32.3.12 UART Input Port Register (UIP*n*)

The UIP*n* registers show the current state of the $\overline{\mathrm{U}n\mathrm{CTS}}$ input.

Address: 0xFC06_0034 (UIP0)
0xFC06_4034 (UIP1)
0xFC06_8034 (UIP2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	1	1	1	1	1	1	1	CTS
W								
Reset:	1	1	1	1	1	1	1	1

Figure 32-15. UART Input Port Registers (UIP*n*)

Table 32-11. UIP*n* Field Descriptions

Field	Description
7–1	Reserved
0 CTS	Current state of clear-to-send. The $\overline{\text{U}n\text{CTS}}$ value is latched and reflects the state of the input pin when UIP*n* is read. **Note:** This bit has the same function and value as UIPCR*n*[CTS]. 0 The current state of the $\overline{\text{U}n\text{CTS}}$ input is logic 0. 1 The current state of the $\overline{\text{U}n\text{CTS}}$ input is logic 1.

32.3.13 UART Output Port Command Registers (UOP1*n*/UOP0*n*)

The $\overline{\text{U}n\text{RTS}}$ output can be asserted by writing a 1 to UOP1*n*[RTS] and negated by writing a 1 to UOP0*n*[RTS].

Address: 0xFC06_0038 (UOP10)
0xFC06_003C (UOP00)
0xFC06_4038 (UOP11)
0xFC06_403C (UOP01)
0xFC06_8038 (UOP12)
0xFC06_803C (UOP02)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	RTS
Reset:	0	0	0	0	0	0	0	0

Figure 32-16. UART Output Port Command Registers (UOP1*n*/UOP0*n*)

Table 32-12. UOP1*n*/UOP0*n* Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 RTS	Output port output. Controls assertion (UOP1)/negation (UOP0) of $\overline{\text{U}n\text{RTS}}$ output. 0 Not affected. 1 Asserts $\overline{\text{U}n\text{RTS}}$ in UOP1. Negates $\overline{\text{U}n\text{RTS}}$ in UOP0.

32.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

32.4.1 Transmitter/Receiver Clock Source

The internal bus clock serves as the basic timing reference for the clock source generator logic, which consists of a clock generator and a programmable 16-bit divider dedicated to each UART. The 16-bit divider is used to produce standard UART baud rates.

32.4.1.1 Programmable Divider

As Figure 32-17 shows, the UART*n* transmitter and receiver can use the following clock sources:

- An external clock signal on the DT*n*IN pin. When not divided, DT*n*IN provides a synchronous clock; when divided by 16, it is asynchronous.
- The internal bus clock supplies an asynchronous clock source divided by 32 and then divided by the 16-bit value programmed in UBG1*n* and UBG2*n*. See Section 32.3.11, "UART Baud Rate Generator Registers (UBG1n/UBG2n)."

The choice of DTIN or internal bus clock is programmed in the UCSR.

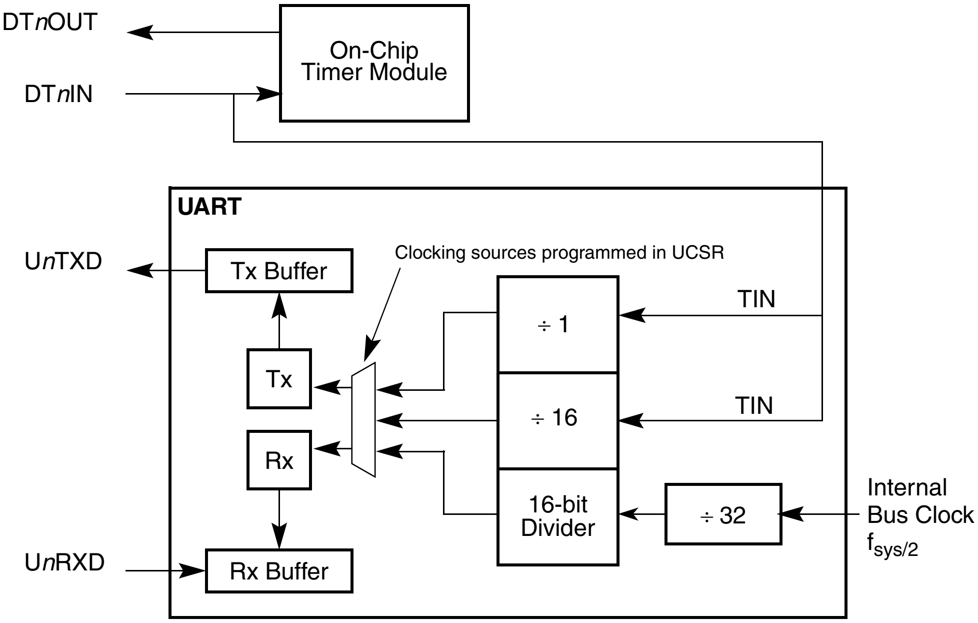


Figure 32-17. Clocking Source Diagram

NOTE

If DT*n*IN is a clocking source for the timer or UART, that timer module cannot use DT*n*IN for timer input capture.

32.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

32.4.1.2.1 Internal Bus Clock Baud Rates

When the internal bus clock is the UART clocking source, it goes through a divide-by-32 prescaler and then passes through the 16-bit divider of the concatenated UBG1*n* and UBG2*n* registers. The baud-rate calculation is:

Using a 133-MHz internal bus clock and letting baud rate equal 9600, then

$$\text{Divider} = \frac{133\text{MHz}}{[32 \times 9600]} = 433(\text{decimal}) = 0\text{x}01\text{B}0(\text{hexadecimal}) \qquad \textbf{\textit{Eqn. 32-1}}$$

Therefore, UBG1*n* equals 0x01 and UBG2*n* equals 0xB0.

32.4.1.2.2 External Clock

An external source clock (DT*n*IN) passes through a divide-by-1 or 16 prescaler. If f_{extc} is the external clock frequency, baud rate can be described with this equation:

$$\text{Baudrate} = \frac{f_{extc}}{(16 \text{ or } 1)} \qquad \textit{Eqn. 32-2}$$

32.4.2 Transmitter and Receiver Operating Modes

Figure 32-18 is a functional block diagram of the transmitter and receiver showing the command and operating registers, which are described generally in the following sections. For detailed descriptions, refer to Section 32.3, "Memory Map/Register Definition."

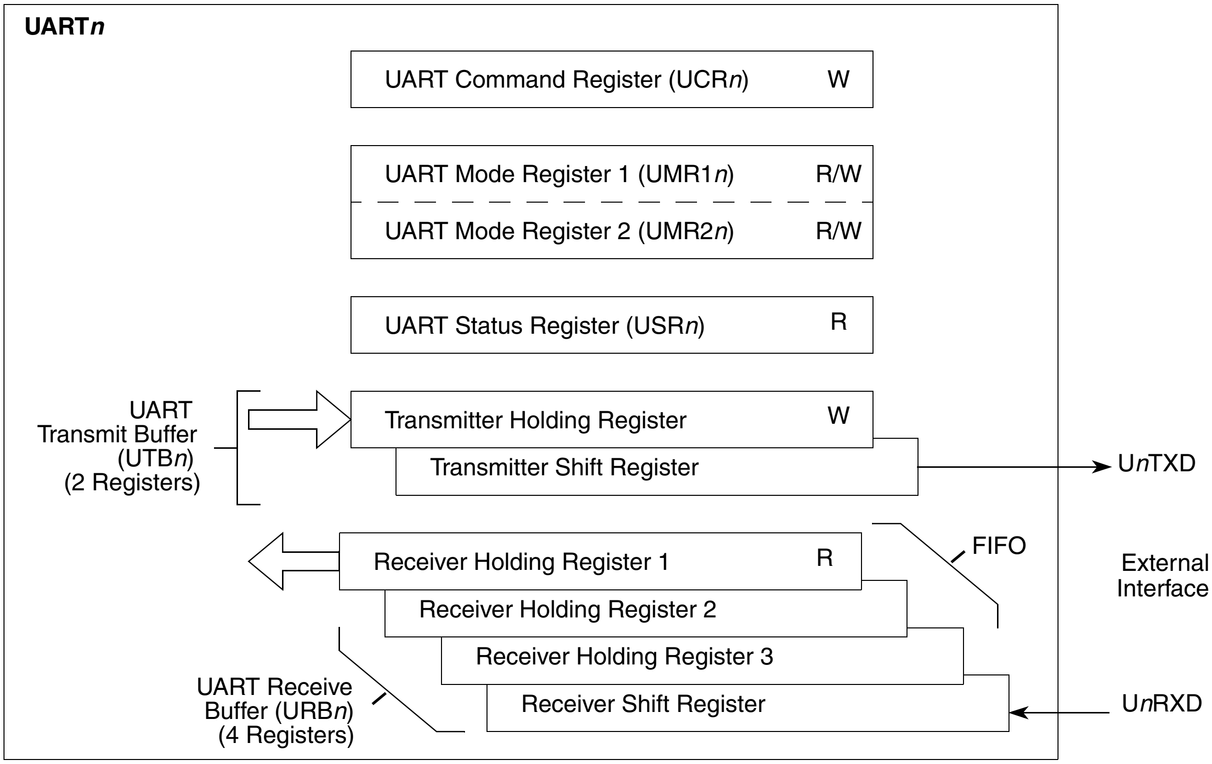

UART*n*
UART Command Register (UCR*n*) W
UART Mode Register 1 (UMR1*n*) R/W
UART Mode Register 2 (UMR2*n*) R/W
UART Status Register (USR*n*) R
UART Transmit Buffer (UTB*n*) (2 Registers)
Transmitter Holding Register W
Transmitter Shift Register
U*n*TXD
Receiver Holding Register 1 R
Receiver Holding Register 2
Receiver Holding Register 3
Receiver Shift Register
FIFO
External Interface
UART Receive Buffer (URB*n*) (4 Registers)
U*n*RXD

Figure 32-18. Transmitter and Receiver Functional Diagram

32.4.2.1 Transmitter

The transmitter is enabled through the UART command register (UCR*n*). When it is ready to accept a character, UART sets USR*n*[TXRDY]. The transmitter converts parallel data from the CPU to a serial bit stream on U*n*TXD. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The lsb is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

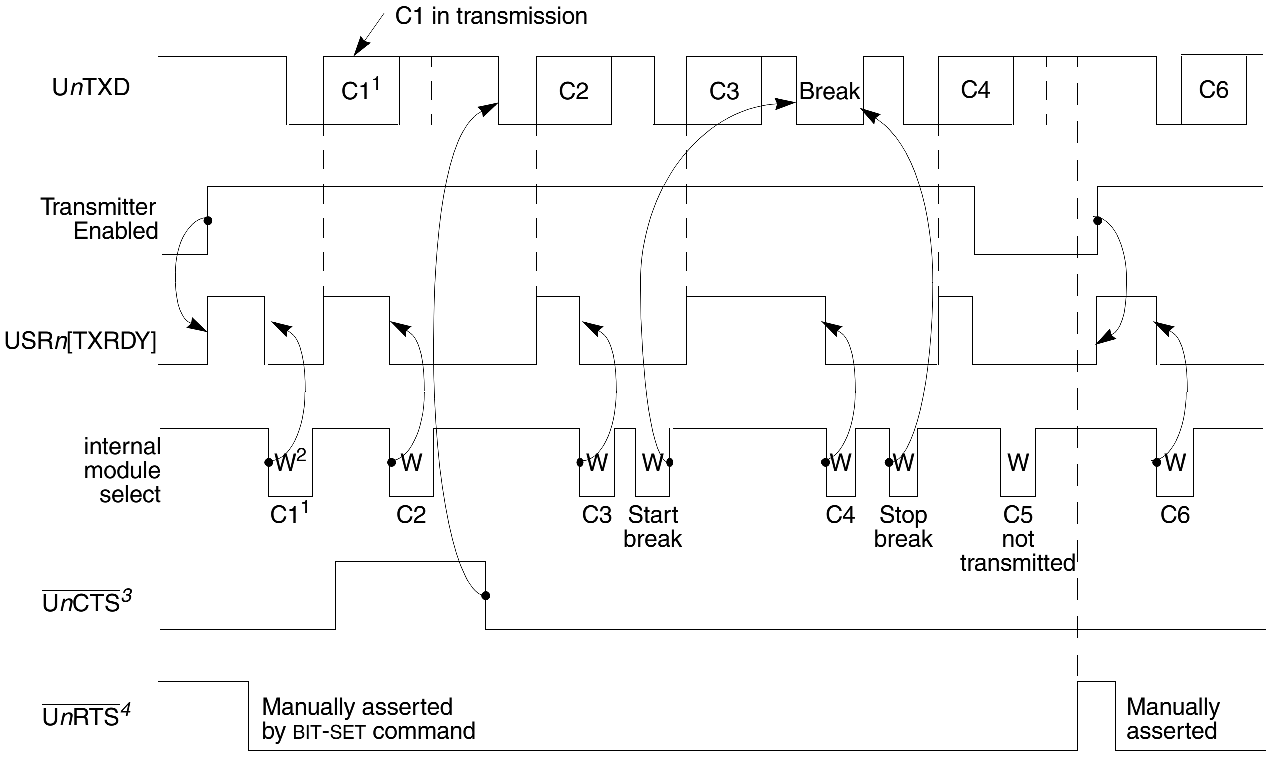
After the stop bits are sent, if no new character is in the transmitter holding register, the U*n*TXD output remains high (mark condition) and the transmitter empty bit (USR*n*[TXEMP]) is set. Transmission resumes and TXEMP is cleared when the CPU loads a new character into the UART transmit buffer (UTB*n*). If the transmitter receives a disable command, it continues until any character in the transmitter shift register is completely sent.

If the transmitter is reset through a software command, operation stops immediately (see Section 32.3.5, "UART Command Registers (UCRn)"). The transmitter is reenabled through the UCR*n* to resume operation after a disable or software reset.

If the clear-to-send operation is enabled, $\overline{\text{U}n\text{CTS}}$ must be asserted for the character to be transmitted. If $\overline{\text{U}n\text{CTS}}$ is negated in the middle of a transmission, the character in the shift register is sent and U*n*TXD remains in mark state until $\overline{\text{U}n\text{CTS}}$ is reasserted. If transmitter is forced to send a continuous low condition by issuing a SEND BREAK command, transmitter ignores the state of $\overline{\text{U}n\text{CTS}}$.

If the transmitter is programmed to automatically negate $\overline{\text{U}n\text{RTS}}$ when a message transmission completes, $\overline{\text{U}n\text{RTS}}$ must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and $\overline{\text{U}n\text{RTS}}$ is appropriately programmed, $\overline{\text{U}n\text{RTS}}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting $\overline{\text{U}n\text{RTS}}$ before the next message is sent.

Figure 32-19 shows the functional timing information for the transmitter.



[1] C*n* = transmit characters
[2] W = write
[3] UMR2*n*[TXCTS] = 1
[4] UMR2*n*[TXRTS] = 1

Figure 32-19. Transmitter Timing Diagram

32.4.2.2 Receiver

The receiver is enabled through its UCR*n*, as described in Section 32.3.5, "UART Command Registers (UCRn)."

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on U*n*RXD, the state of U*n*RXD is sampled eight times on the edge of the bit time clock starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If U*n*RXD is sampled high, start bit is invalid and the search for the valid start bit begins again.

If U*n*RXD remains low, a valid start bit is assumed. The receiver continues sampling the input at one-bit time intervals at the theoretical center of the bit until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the U*n*RXD input is sampled on the rising edge of the programmed clock source. The lsb is received first. The data then transfers to a receiver holding register and USR*n*[RXRDY] is set. If the character is less than 8 bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, receiver immediately looks for the next start bit. However, if a non-zero character is received without a stop bit (framing error) and U*n*RXD remains low for one-half of the bit period after the stop bit is sampled, receiver operates as if a new start bit were detected. Parity error, framing error, overrun error, and received break conditions set the respective PE, FE, OE, and RB error and break flags in the USR*n* at the received character boundary. They are valid only if USR*n*[RXRDY] is set.

If a break condition is detected (U*n*RXD is low for the entire character including the stop bit), a character of all 0s loads into the receiver holding register and USR*n*[RB,RXRDY] are set. U*n*RXD must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. The receiver places the damaged character in the Rx FIFO and sets the corresponding USR*n* error bits and USR*n*[RXRDY]. Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets USR*n*[RB,RXRDY].

Figure 32-20 shows receiver functional timing.

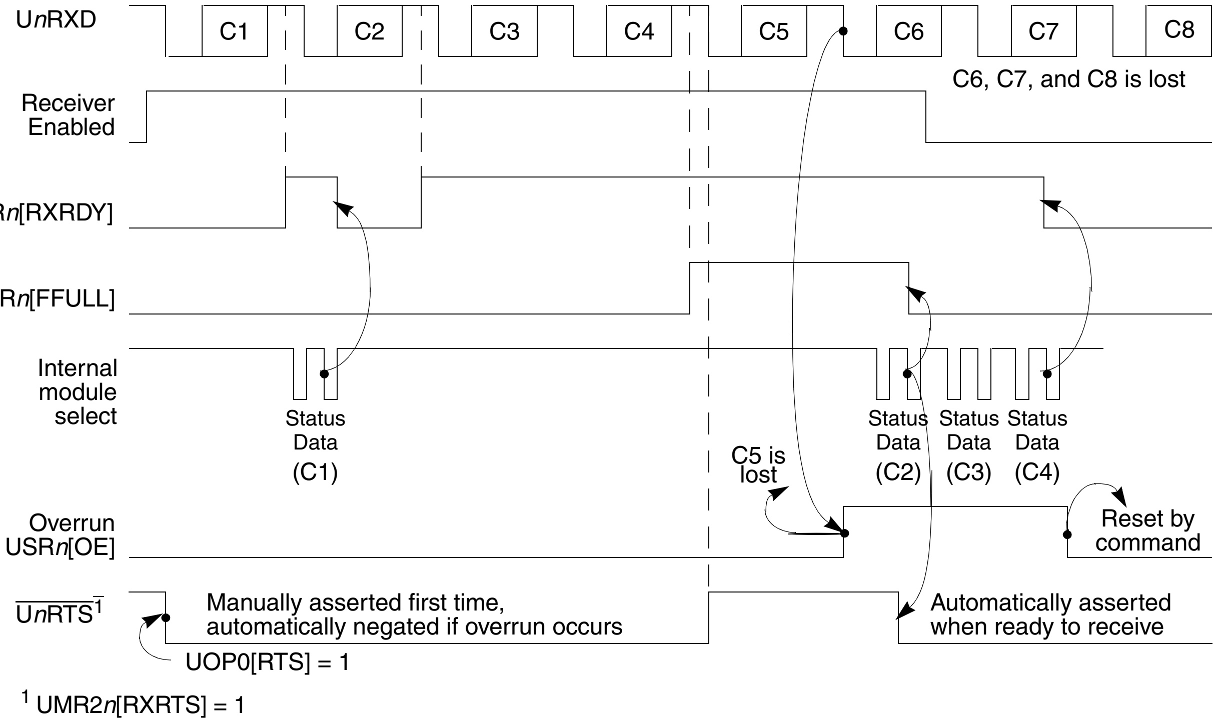


Figure 32-20. Receiver Timing Diagram

32.4.2.3 FIFO

The FIFO is used in the UART's receive buffer logic. The FIFO consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the U*n*RXD (see Figure 32-18). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Therefore, data flowing from the receiver to the CPU is quadruple-buffered.

In addition to the data byte, three status bits—parity error (PE), framing error (FE), and received break (RB)—are appended to each data character in the FIFO; overrun error (OE) is not appended. By programming the ERR bit in the UART's mode register (UMR1*n*), status is provided in character or block modes.

USR*n*[RXRDY] is set when at least one character is available to be read by the CPU. A read of the receive buffer produces an output of data from the top of the FIFO. After the read cycle, the data at the top of the FIFO and its associated status bits are popped and the receiver shift register can add new data at the bottom of the FIFO. The FIFO-full status bit (FFULL) is set if all three positions are filled with data. The RXRDY or FFULL bit can be selected to cause an interrupt and TXRDY or RXRDY can be used to generate a DMA request.

The two error modes are selected by UMR1*n*[ERR]:

- In character mode (UMR1*n*[ERR] = 0), status is given in the USR*n* for the character at the top of the FIFO.

- In block mode, the USR*n* shows a logical OR of all characters reaching the top of the FIFO since the last RESET ERROR STATUS command. Status is updated as characters reach the top of the FIFO. Block mode offers a data-reception speed advantage where the software overhead of error-checking each character cannot be tolerated. However, errors are not detected until the check is performed at the end of an entire message—the faulting character is not identified.

In either mode, reading the USR*n* does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The USR*n* should be read before reading the receive buffer. If all three receiver holding registers are full, a new character is held in the receiver shift register until space is available. However, if a second new character is received, the contents of the character in the receiver shift register is lost, the FIFOs are unaffected, and USR*n*[OE] is set when the receiver detects the start bit of the new overrunning character.

To support flow control, the receiver can be programmed to automatically negate and assert $\overline{\text{U}n\text{RTS}}$, in which case the receiver automatically negates $\overline{\text{U}n\text{RTS}}$ when a valid start bit is detected and the FIFO is full. The receiver asserts $\overline{\text{U}n\text{RTS}}$ when a FIFO position becomes available; therefore, connecting $\overline{\text{U}n\text{RTS}}$ to the $\overline{\text{U}n\text{CTS}}$ input of the transmitting device can prevent overrun errors.

NOTE

> The receiver continues reading characters in the FIFO if the receiver is disabled. If the receiver is reset, the FIFO, $\overline{\text{U}n\text{RTS}}$ control, all receiver status bits, interrupts, and DMA requests are reset. No more characters are received until the receiver is reenabled.

32.4.3 Looping Modes

The UART can be configured to operate in various looping modes. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs and in Section 32.3, “Memory Map/Register Definition.”

The UART’s transmitter and receiver should be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

32.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in Figure 32-21, the UART automatically resends received data bit by bit. The local CPU-to-receiver communication continues normally, but the CPU-to-transmitter link is disabled. In this mode, received data is clocked on the receiver clock and re-sent on U*n*TXD. The receiver must be enabled, but the transmitter need not be.

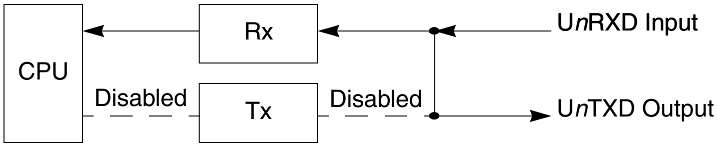


Figure 32-21. Automatic Echo

Because the transmitter is inactive, USR*n*[TXEMP,TXRDY] is inactive and data is sent as it is received. Received parity is checked but not recalculated for transmission. Character framing is also checked, but stop bits are sent as they are received. A received break is echoed as received until the next valid start bit is detected.

32.4.3.2 Local Loopback Mode

Figure 32-22 shows how U*n*TXD and U*n*RXD are internally connected in local loopback mode. This mode is for testing the operation of a UART by sending data to the transmitter and checking data assembled by the receiver to ensure proper operations.

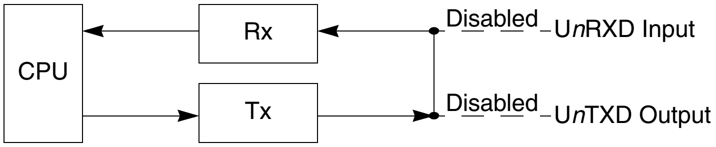


Figure 32-22. Local Loopback

Features of this local loopback mode are:

- Transmitter and CPU-to-receiver communications continue normally in this mode.
- U*n*RXD input data is ignored.
- U*n*TXD is held marking.
- The receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be.

32.4.3.3 Remote Loopback Mode

In remote loopback mode, shown in Figure 32-23, the UART automatically transmits received data bit by bit on the U*n*TXD output. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote UART. For this mode, transmitter uses the receiver clock.

Because the receiver is not active, received data cannot be read by the CPU and all status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are sent as they are received. A received break is echoed as received until next valid start bit is detected.

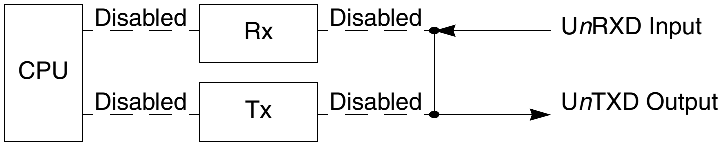


Figure 32-23. Remote Loopback

32.4.4 Multidrop Mode

Setting UMR1*n*[PM] programs the UART to operate in a wake-up mode for multidrop or multiprocessor applications. In this mode, a master can transmit an address character followed by a block of data characters targeted for one of up to 256 slave stations.

Although slave stations have their receivers disabled, they continuously monitor the master's data stream. When the master sends an address character, the slave receiver notifies its respective CPU by setting USR*n*[RXRDY] and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Unaddressed slave stations continue monitoring the data stream. Data fields in the data stream are separated by an address character. After a slave receives a block of data, its CPU disables the receiver and repeats the process. Functional timing information for multidrop mode is shown in Figure 32-24.

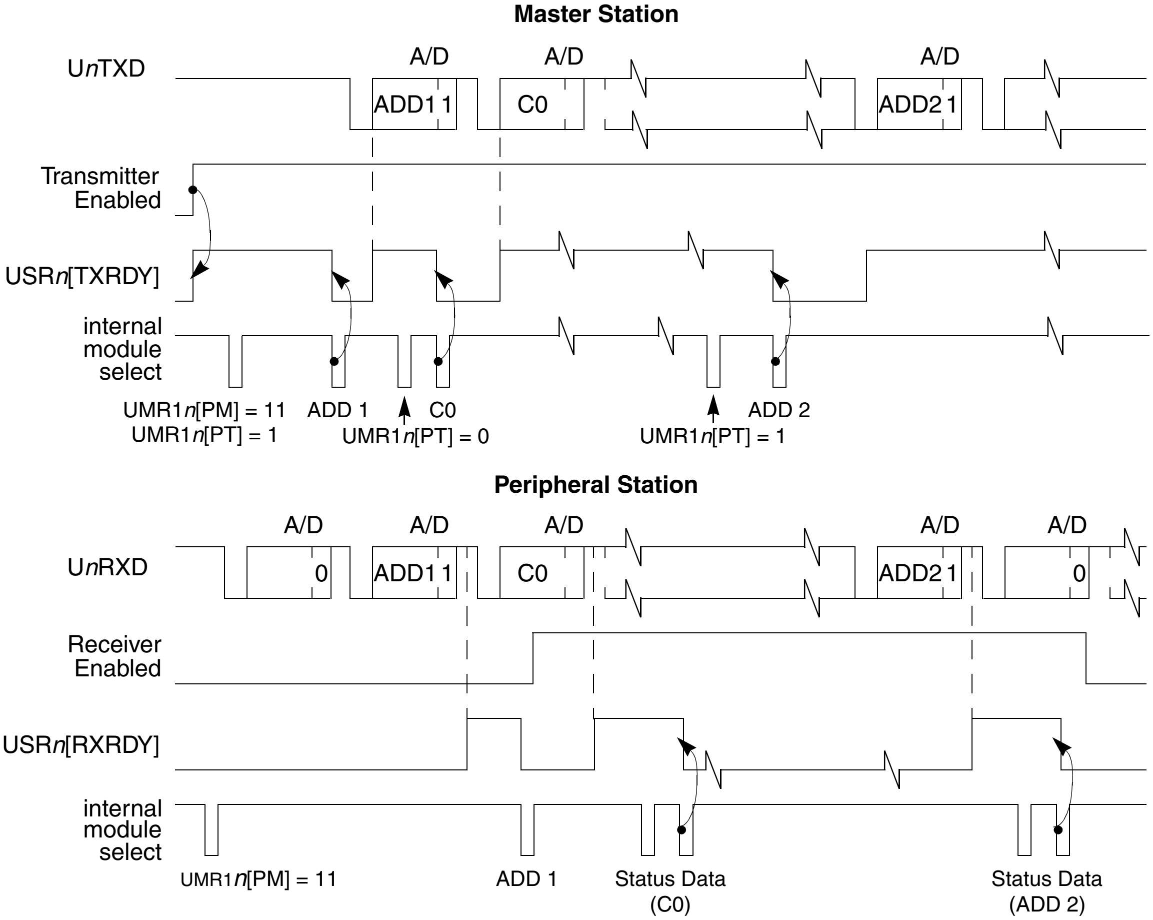


Figure 32-24. Multidrop Mode Timing Diagram

A character sent from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. A/D equals 1 indicates an address character; A/D equals 0 indicates a data character. The polarity of A/D is selected through UMR1*n*[PT]. UMR1*n* should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RXRDY bit and loads the character into the receiver holding register FIFO provided the received A/D bit is a 1 (address tag). The character is discarded if the received A/D bit is 0 (data tag). If the receiver is enabled, all received characters are transferred to the CPU through the receiver holding register during read operations.

In either case, data bits load into the data portion of the FIFO while the A/D bit loads into the status portion of the FIFO normally used for a parity error (USR*n*[PE]).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may continues containing error detection and correction information. If 8-bit characters are not required, one way to provide error detection is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

32.4.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

32.4.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

32.4.5.2 Write Cycles

The UART module accepts write data as bytes only. Write cycles to read-only or reserved registers complete normally without an error termination, but data is ignored.

32.5 Initialization/Application Information

The software flowchart, Figure 32-25, consists of:

- UART module initialization—These routines consist of SINIT and CHCHK (See Sheet 1 p. 32-28 and Sheet 2 p. 32-29). Before SINIT is called at system initialization, the calling routine allocates 2 words on the system FIFO. On return to the calling routine, SINIT passes UART status data on the FIFO. If SINIT finds no errors, the transmitter and receiver are enabled. SINIT calls CHCHK to perform the checks. When called, SINIT places the UART in local loopback mode and checks for the following errors:
  - Transmitter never ready
  - Receiver never ready
  - Parity error
  - Incorrect character received
- I/O driver routine—This routine (See Sheet 4 p. 32-31 and Sheet 5 p. 32-32) consists of INCH, the terminal input character routine which gets a character from the receiver, and OUTCH, which sends a character to the transmitter.
- Interrupt handling—This consists of SIRQ (See Sheet 4 p. 32-31), which is executed after the UART module generates an interrupt caused by a change-in-break (beginning of a break). SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

32.5.1 Interrupt and DMA Request Initialization

32.5.1.1 Setting up the UART to Generate Core Interrupts

The list below provides steps to properly initialize the UART to generate an interrupt request to the processor's interrupt controller. See Section 17.2.9.1, "Interrupt Sources," for details on interrupt assignments for the UART modules.

1. Initialize the appropriate ICR*x* register in the interrupt controller.
2. Unmask appropriate bits in IMR in the interrupt controller.
3. Unmask appropriate bits in the core's status register (SR) to enable interrupts.
4. If TXRDY or RXRDY generates interrupt requests, verify that the corresponding UART DMA channels are not enabled.
5. Initialize interrupts in the UART, see Table 32-13.

Table 32-13. UART Interrupts

Register	Bit	Interrupt
UMR1*n*	6	RxIRQ
UIMR*n*	7	Change of State (COS)
UIMR*n*	2	Delta Break
UIMR*n*	1	RxFIFO Full
UIMR*n*	0	TXRDY

32.5.1.2 Setting up the UART to Request DMA Service

The UART is capable of generating two internal DMA request signals: transmit and receive.

The transmit DMA request signal is asserted when the TXRDY (transmitter ready) in the UART interrupt status register (UISR*n*[TXRDY]) is set. When the transmit DMA request signal is asserted, the DMA can initiate a data copy, reading the next character transmitted from memory and writing it into the UART transmit buffer (UTB*n*). This allows the DMA channel to stream data from memory to the UART for transmission without processor intervention. After the entire message has been moved into the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) could query the UART programming model to determine the end-of-transmission status.

Similarly, the receive DMA request signal is asserted when the FIFO full or receive ready (FFULL/RXRDY) flag in the interrupt status register (UISR*n*[FFULL/RXRDY]) is set. When the receive DMA request signal is asserted, the DMA can initiate a data move, reading the appropriate characters from the UART receive buffer (URB*n*) and storing them in memory. This allows the DMA channel to stream data from the UART receive buffer into memory without processor intervention. After the entire message has been moved from the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) should query the UART programming model to determine the end-of-transmission status. In typical applications, the receive DMA request should be configured to use RXRDY directly (and not FFULL) to remove any complications related to retrieving the final characters from the FIFO buffer.

The implementation described in this section allows independent DMA processing of transmit and receive data while continuing to support interrupt notification to the processor for $\overline{\text{CTS}}$ change-of-state and delta break error managing.

Table 32-14 shows the DMA requests.

Table 32-14. UART DMA Requests

Register	Bit	DMA Request
UISR*n*	1	Receive DMA request
UISR*n*	0	Transmit DMA request

32.5.2 UART Module Initialization Sequence

The following shows the UART module initialization sequence.

1. UCR*n*:
   a) Reset the receiver and transmitter.
   b) Reset the mode pointer (MISC[2–0] = 0b001).
2. UIMR*n*: Enable the desired interrupt sources.
3. UACR*n*: Initialize the input enable control (IEC bit).
4. UCSR*n*: Select the receiver and transmitter clock. Use timer as source if required.
5. UMR1*n*:
   a) If preferred, program operation of receiver ready-to-send (RXRTS bit).
   a) Select receiver-ready or FIFO-full notification (RXRDY/FFULL bit).
   b) Select character or block error mode (ERR bit).
   c) Select parity mode and type (PM and PT bits).
   d) Select number of bits per character (B/Cx bits).
6. UMR2*n*:
   a) Select the mode of operation (CM bits).
   b) If preferred, program operation of transmitter ready-to-send (TXRTS).
   c) If preferred, program operation of clear-to-send (TXCTS bit).
   d) Select stop-bit length (SB bits).
7. UCR*n*: Enable transmitter and/or receiver.

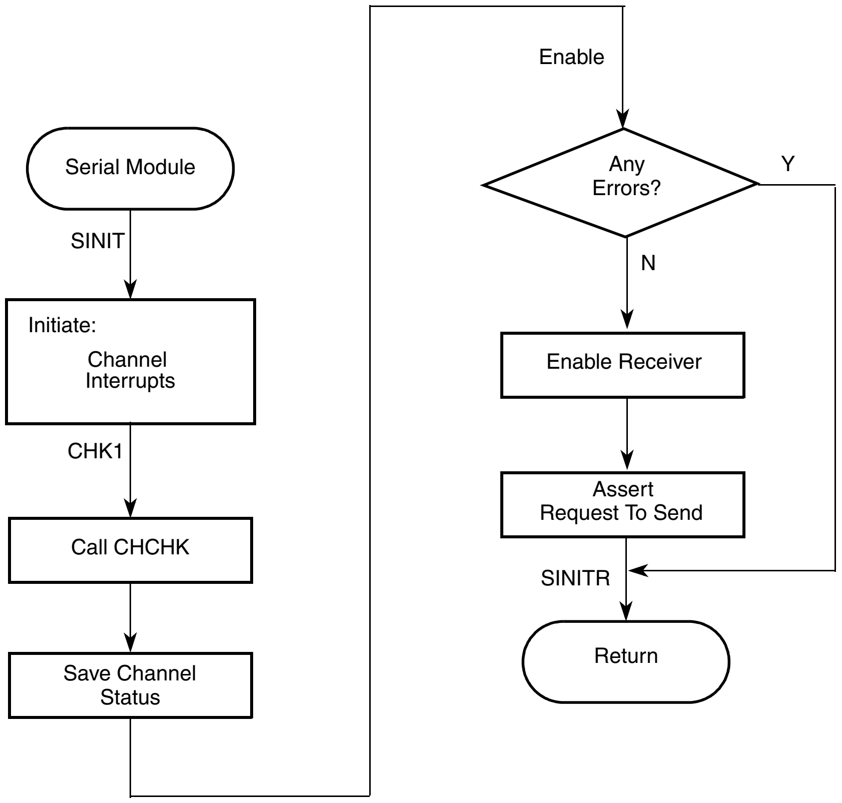


Figure 32-25. UART Mode Programming Flowchart (Sheet 1 of 5)

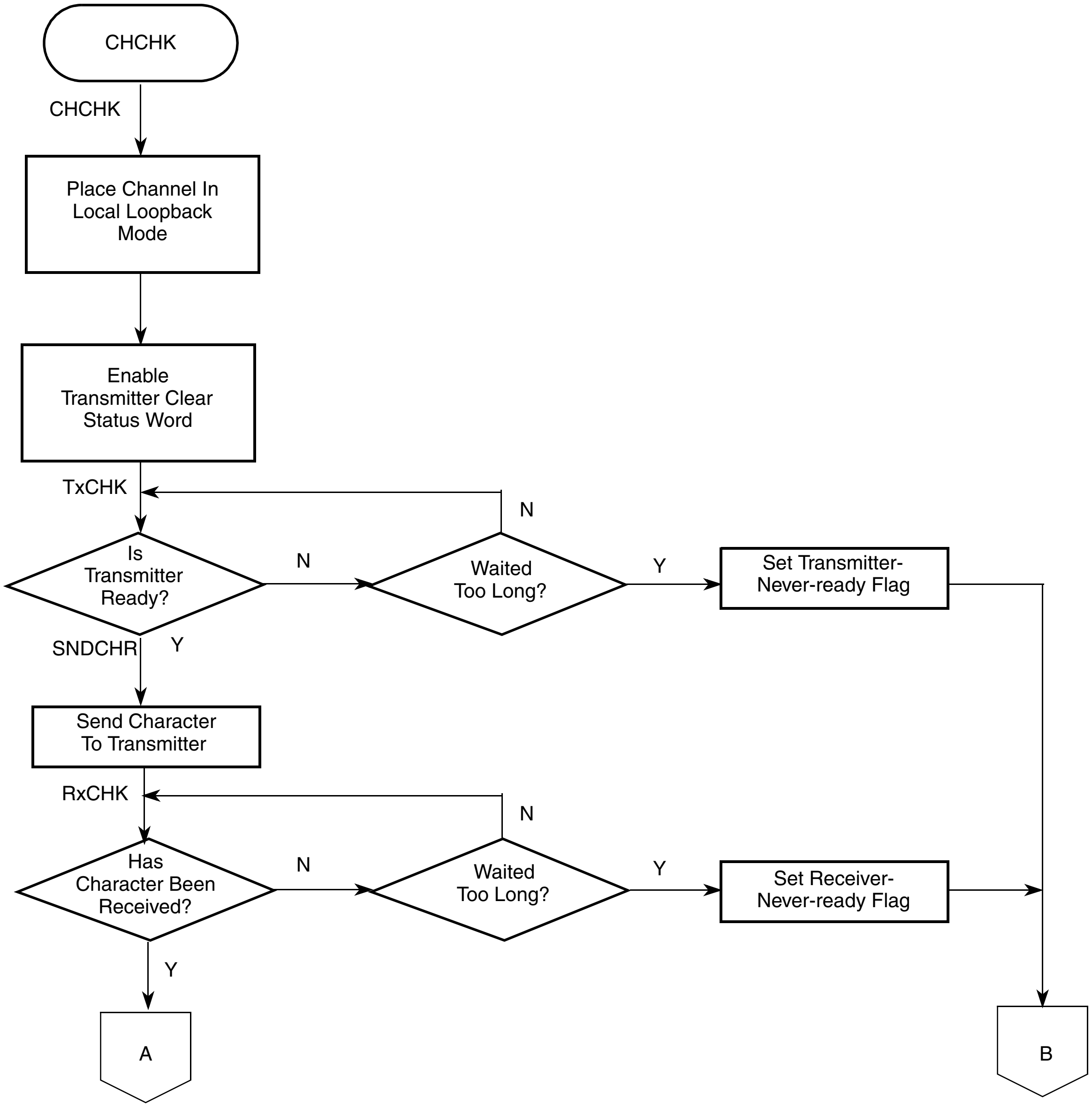

CHCHK
CHCHK
Place Channel In Local Loopback Mode
Enable Transmitter Clear Status Word
TxCHK
Is Transmitter Ready?
N
Waited Too Long?
N
Y
Set Transmitter-Never-ready Flag
SNDCHR
Y
Send Character To Transmitter
RxCHK
Has Character Been Received?
N
Waited Too Long?
N
Y
Set Receiver-Never-ready Flag
Y
A
B

Figure 32-25. UART Mode Programming Flowchart (Sheet 2 of 5)

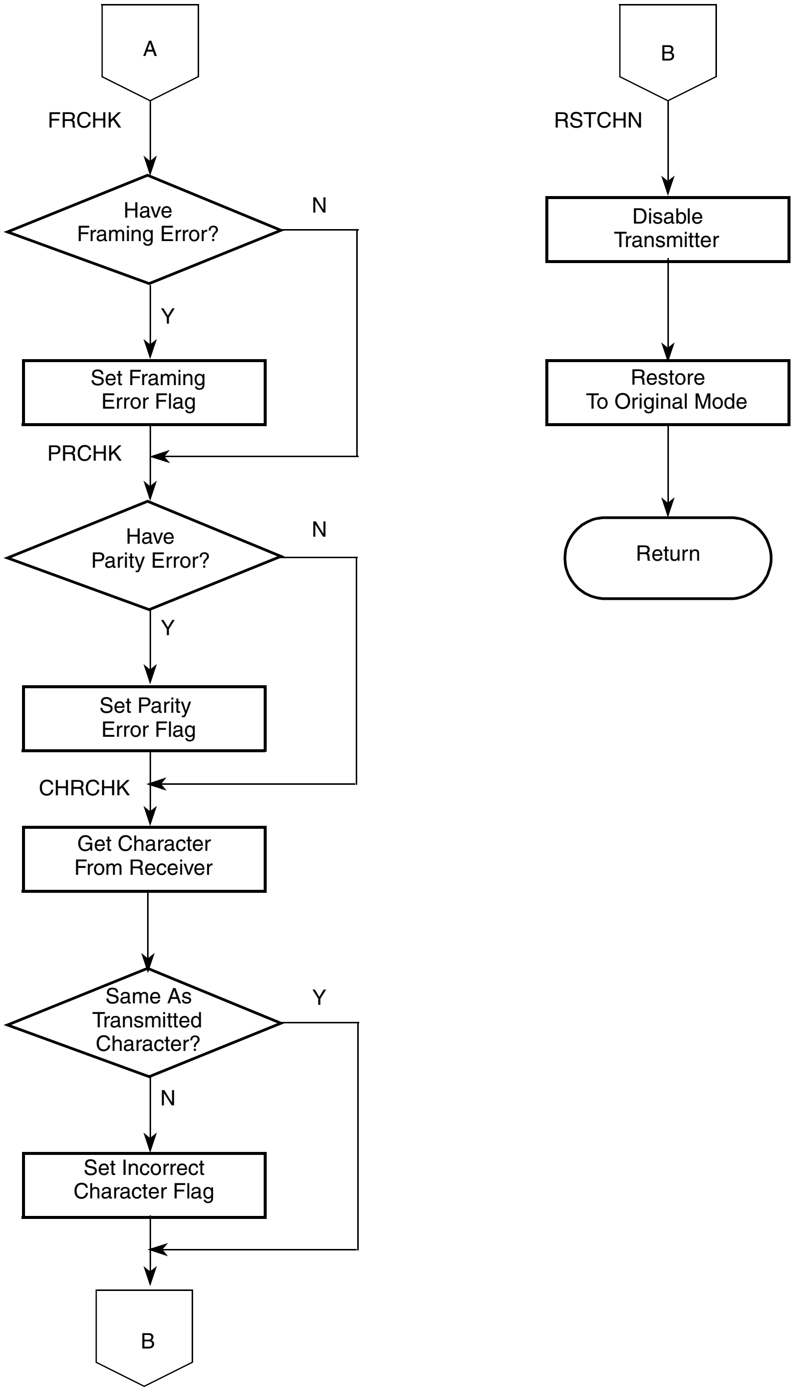


Figure 32-25. UART Mode Programming Flowchart (Sheet 3 of 5)

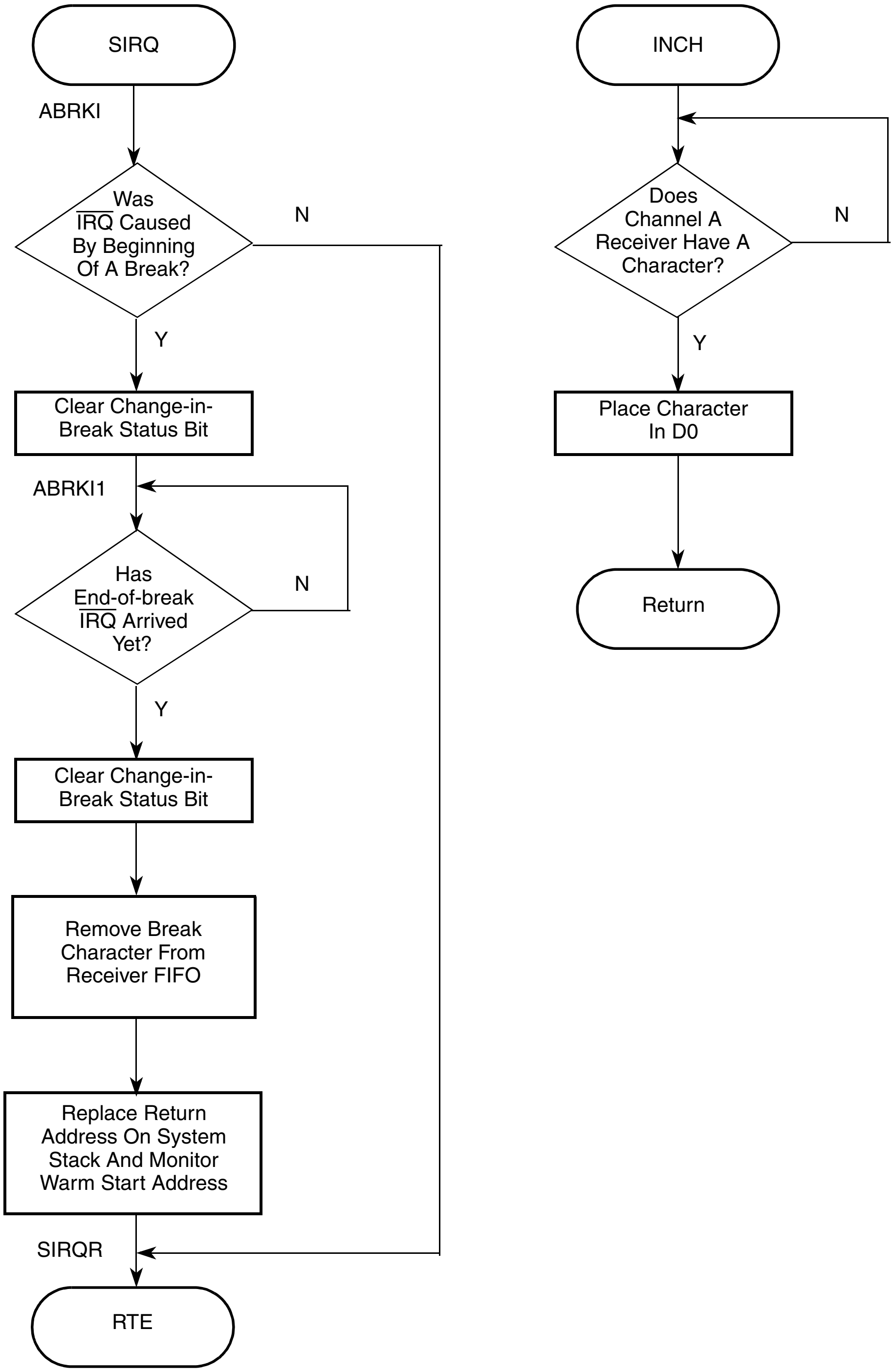

SIRQ
ABRKI
Was $\overline{\text{IRQ}}$ Caused By Beginning Of A Break?
N
Y
Clear Change-in-Break Status Bit
ABRKI1
Has End-of-break $\overline{\text{IRQ}}$ Arrived Yet?
N
Y
Clear Change-in-Break Status Bit
Remove Break Character From Receiver FIFO
Replace Return Address On System Stack And Monitor Warm Start Address
SIRQR
RTE
INCH
Does Channel A Receiver Have A Character?
N
Y
Place Character In D0
Return

Figure 32-25. UART Mode Programming Flowchart (Sheet 4 of 5)

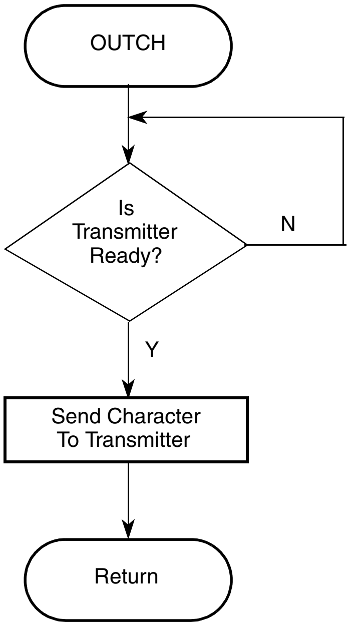


Figure 32-25. UART Mode Programming Flowchart (Sheet 5 of 5)

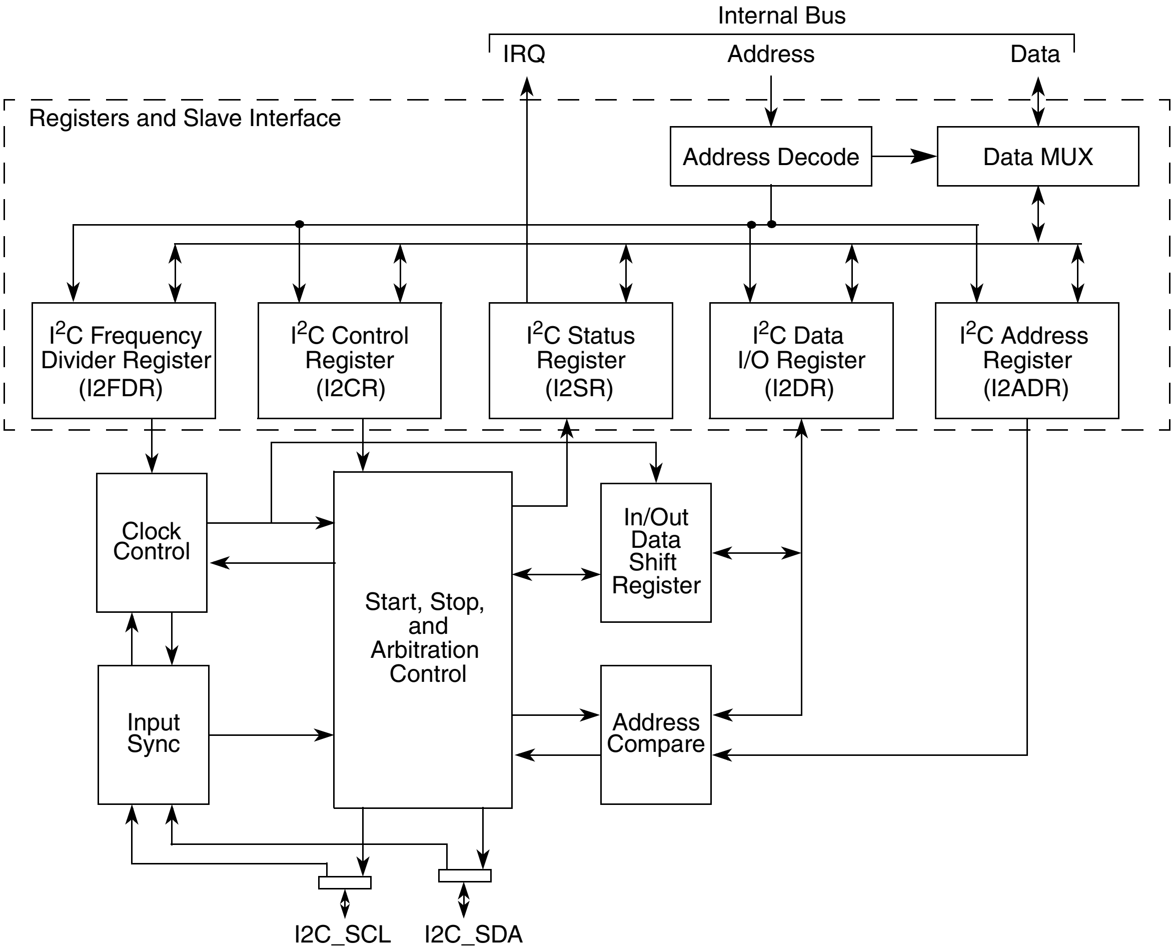
Chapter 33
I²C Interface

33.1 Introduction

This chapter describes the I^{2}C module, clock synchronization, and I^{2}C programming model registers. It also provides extensive programming examples.

33.1.1 Block Diagram

Figure 33-1 is a I^{2}C module block diagram, illustrating the interaction of the registers described in Section 33.2, "Memory Map/Register Definition".



Internal Bus
IRQ
Address
Data
Registers and Slave Interface
Address Decode
Data MUX
I²C Frequency Divider Register (I2FDR)
I²C Control Register (I2CR)
I²C Status Register (I2SR)
I²C Data I/O Register (I2DR)
I²C Address Register (I2ADR)
Clock Control
Start, Stop, and Arbitration Control
In/Out Data Shift Register
Input Sync
Address Compare
I2C_SCL
I2C_SDA

Figure 33-1. I^{2}C Module Block Diagram

33.1.2 Overview

I^{2}C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications that require occasional communication between many devices over a short distance. The flexible I^{2}C bus allows additional devices to connect to the bus for expansion and system development.

The interface operates up to 100 Kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of the internal bus clock divided by 20, with reduced bus loading. The maximum communication length and the number of devices connected are limited by a maximum bus capacitance of 400 pF.

The I^{2}C system is a true multiple-master bus; it uses arbitration and collision detection to prevent data corruption in the event that multiple devices attempt to control the bus simultaneously. This supports complex applications with multiprocessor control and can be used for rapid testing and alignment of end products through external connections to an assembly-line computer.

NOTE

The I^{2}C module is compatible with the Philips I^{2}C bus protocol. For information on system configuration, protocol, and restrictions, see *The I^{2}C Bus Specification, Version 2.1*.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 16, “Pin Multiplexing and Control”) prior to configuring the I^{2}C module.

33.1.3 Features

The I^{2}C module has these key features:

- Compatibility with I^{2}C bus standard version 2.1
- Multiple-master operation
- Software-programmable for one of 50 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

33.2 Memory Map/Register Definition

The below table lists the configuration registers used in the I^{2}C interface.

Table 33-1. I^{2}C Module Memory Map

Address	Register	Access	Reset Value	Section/Page
0xFC05_8000	I^{2}C Address Register (I2ADR)	R/W	0x00	33.2.1/33-3
0xFC05_8004	I^{2}C Frequency Divider Register (I2FDR)	R/W	0x00	33.2.2/33-3
0xFC05_8008	I^{2}C Control Register (I2CR)	R/W	0x00	33.2.3/33-4
0xFC05_800C	I^{2}C Status Register (I2SR)	R/W	0x81	33.2.4/33-5
0xFC05_8010	I^{2}C Data I/O Register (I2DR)	R/W	0x00	33.2.5/33-6

33.2.1 I^{2}C Address Register (I2ADR)

I2ADR holds the address the I^{2}C responds to when addressed as a slave. It is not the address sent on the bus during the address transfer when the module is performing a master transfer.

Address: 0xFC05_8000 (I2ADR) Access: User read/write

	7	6	5	4	3	2	1	0
R	ADR							0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 33-2. I^{2}C Address Register (I2ADR)

Table 33-2. I2ADR Field Descriptions

Field	Description
7–1 ADR	Slave address. Contains the specific slave address to be used by the I^{2}C module. Slave mode is the default I^{2}C mode for an address match on the bus.
0	Reserved, must be cleared.

33.2.2 I^{2}C Frequency Divider Register (I2FDR)

The I2FDR, shown in Figure 33-3, provides a programmable prescaler to configure the I^{2}C clock for bit-rate selection.

Address: 0xFC05_8004 (I2FDR) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	IC					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 33-3. I^{2}C Frequency Divider Register (I2FDR)

Table 33-3. I2FDR Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 IC	I^{2}C clock rate. Prescales the clock for bit-rate selection. The serial bit clock frequency is equal to the internal bus clock divided by the divider shown below. Due to potentially slow I2C_SCL and I2C_SDA rise and fall times, bus signals are sampled at the prescaler frequency.

IC	Divider	IC	Divider	IC	Divider	IC	Divider
0x00	28	0x10	288	0x20	20	0x30	160
0x01	30	0x11	320	0x21	22	0x31	192
0x02	34	0x12	384	0x22	24	0x32	224
0x03	40	0x13	480	0x23	26	0x33	256
0x04	44	0x14	576	0x24	28	0x34	320
0x05	48	0x15	640	0x25	32	0x35	384
0x06	56	0x16	768	0x26	36	0x36	448
0x07	68	0x17	960	0x27	40	0x37	512
0x08	80	0x18	1152	0x28	48	0x38	640
0x09	88	0x19	1280	0x29	56	0x39	768
0x0A	104	0x1A	1536	0x2A	64	0x3A	896
0x0B	128	0x1B	1920	0x2B	72	0x3B	1024
0x0C	144	0x1C	2304	0x2C	80	0x3C	1280
0x0D	160	0x1D	2560	0x2D	96	0x3D	1536
0x0E	192	0x1E	3072	0x2E	112	0x3E	1792
0x0F	240	0x1F	3840	0x2F	128	0x3F	2048

33.2.3 I^{2}C Control Register (I2CR)

I2CR enables the I^{2}C module and the I^{2}C interrupt. It also contains bits that govern operation as a slave or a master.

Address: 0xFC05_8008 (I2CR)

Access: User read/write

	7	6	5	4	3	2	1	0
R	IEN	IIEN	MSTA	MTX	TXAK	RSTA	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 33-4. I^{2}C Control Register (I2CR)

Table 33-4. I2CR Field Descriptions

Field	Description
7 IEN	I^{2}C enable. Controls the software reset of the entire I^{2}C module. If the module is enabled in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next START condition is detected. Master mode is not aware that the bus is busy; initiating a start cycle may corrupt the current bus cycle, ultimately causing the current master or the I^{2}C module to lose arbitration, after which bus operation returns to normal. 0 The I^{2}C module is disabled, but registers can be accessed. 1 The I^{2}C module is enabled. This bit must be set before any other I2CR bits have any effect.
6 IIEN	I^{2}C interrupt enable. 0 I^{2}C module interrupts are disabled, but currently pending interrupt condition is not cleared. 1 I^{2}C module interrupts are enabled. An I^{2}C interrupt occurs if I2SR[IIF] is also set.
5 MSTA	Master/slave mode select bit. If the master loses arbitration, MSTA is cleared without generating a STOP signal. 0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode. 1 Master mode. Changing MSTA from 0 to 1 signals a START on the bus and selects master mode.
4 MTX	Transmit/receive mode select bit. Selects the direction of master and slave transfers. 0 Receive 1 Transmit. When the device is addressed as a slave, software must set MTX according to I2SR[SRW]. In master mode, MTX must be set according to the type of transfer required. Therefore, when the MCU addresses a slave device, MTX is always 1.
3 TXAK	Transmit acknowledge enable. Specifies the value driven onto I2C_SDA during acknowledge cycles for master and slave receivers. Writing TXAK applies only when the I^{2}C bus is a receiver. 0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (acknowledge bit = 1).
2 RSTA	Repeat start. Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration. 0 No repeat start 1 Generates a repeated START condition.
1–0	Reserved, must be cleared.

33.2.4 I^{2}C Status Register (I2SR)

I2SR contains bits that indicate transaction direction and status.

Address: 0xFC05_800C (I2SR) Access: User read/write

	7	6	5	4	3	2	1	0
R	ICF	IAAS	IBB	IAL	0	SRW	IIF	RXAK
W								
Reset:	1	0	0	0	0	0	0	1

Figure 33-5. I^{2}C Status Register (I2SR)

Table 33-5. I2SR Field Descriptions

Field	Description
7 ICF	I²C Data transferring bit. While one byte of data is transferred, ICF is cleared. 0 Transfer in progress 1 Transfer complete. Set by falling edge of ninth clock of a byte transfer.
6 IAAS	I²C addressed as a slave bit. The CPU is interrupted if I2CR[IIEN] is set. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CR clears this bit. 0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
5 IBB	I²C bus busy bit. Indicates the status of the bus. 0 Bus is idle. If a STOP signal is detected, IBB is cleared. 1 Bus is busy. When START is detected, IBB is set.
4 IAL	I²C arbitration lost. Set by hardware in the following circumstances. (IAL must be cleared by software by writing zero to it.) • I2C_SDA sampled low when the master drives high during an address or data-transmit cycle. • I2C_SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it.
3	Reserved, must be cleared.
2 SRW	Slave read/write. When IAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master. SRW is valid only when a complete transfer has occurred, no other transfers have been initiated, and the I²C module is a slave and has an address match. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
1 IIF	I²C interrupt. Must be cleared by software by writing a 0 in the interrupt routine. 0 No I²C interrupt pending 1 An interrupt is pending, which causes a processor interrupt request (if IIEN = 1). Set when one of the following occurs: • Complete one byte transfer (set at the falling edge of the ninth clock) • Reception of a calling address that matches its own specific address in slave-receive mode • Arbitration lost
0 RXAK	Received acknowledge. The value of I2C_SDA during the acknowledge bit of a bus cycle. 0 An acknowledge signal was received after the completion of 8-bit data transmission on the bus 1 No acknowledge signal was detected at the ninth clock.

33.2.5 I²C Data I/O Register (I2DR)

In master-receive mode, reading I2DR allows a read to occur and for the next data byte to be received. In slave mode, the same function is available after the I²C has received its slave address.

Address: 0xFC05_8010 (I2DR) Access: User read/write

	7	6	5	4	3	2	1	0
R W	DATA							
Reset:	0	0	0	0	0	0	0	0

Figure 33-6. I^{2}C Data I/O Register (I2DR)

Table 33-6. I2DR Field Description

Field	Description
7–0 DATA	I^{2}C data. When data is written to this register in master transmit mode, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates the reception of the next byte of data. In slave mode, the same functions are available after an address match has occurred. **Note:** In master transmit mode, the first byte of data written to I2DR following assertion of I2CR[MSTA] is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/W bit (in position D0). This bit (D0) is not automatically appended by the hardware, software must provide the appropriate R/W bit. **Note:** I2CR[MSTA] generates a start when a master does not already own the bus. I2CR[RSTA] generates a start (restart) without the master first issuing a stop (i.e., the master already owns the bus). To start the read of data, a dummy read to this register starts the read process from the slave. The next read of the I2DR register contains the actual data.

33.3 Functional Description

The I^{2}C module uses a serial data line (I2C_SDA) and a serial clock line (I2C_SCL) for data transfer. For I^{2}C compliance, all devices connected to these two signals must have open drain or open collector outputs. The logic AND function is exercised on both lines with external pull-up resistors.

Out of reset, the I^{2}C default state is as a slave receiver. Therefore, when not programmed to be a master or responding to a slave transmit address, the I^{2}C module should return to the default slave receiver state. See Section 33.4.1, “Initialization Sequence,” for exceptions.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. These are discussed in the following sections.

33.3.1 START Signal

When no other device is bus master (I2C_SCL and I2C_SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in Figure 33-7). A START signal is defined as a high-to-low transition of I2C_SDA while I2C_SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

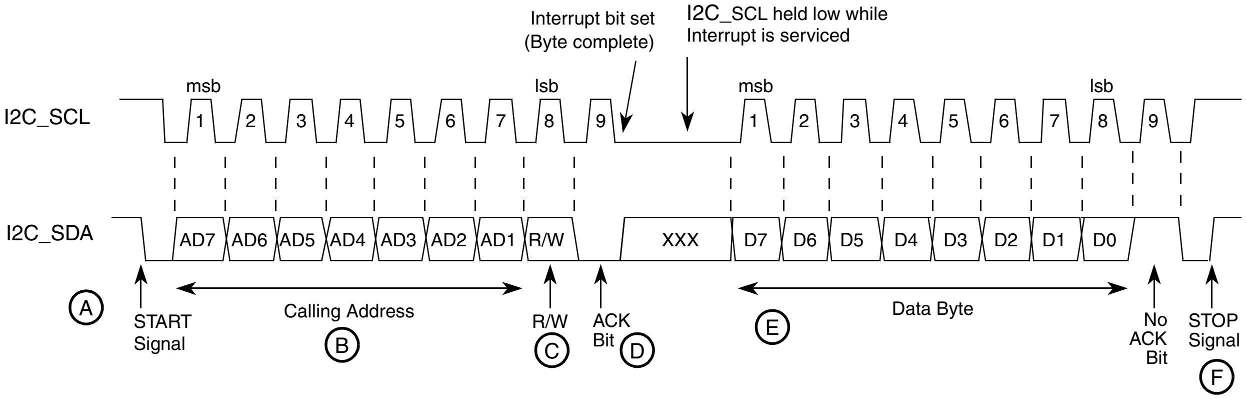


Figure 33-7. I²C Standard Communication Protocol

33.3.2 Slave Address Transmission

The master sends the slave address in the first byte after the START signal (B). After the seven-bit calling address, it sends the R/W bit (C), which tells the slave data transfer direction (0 equals write transfer, 1 equals read transfer).

Each slave must have a unique address. An I²C master must not transmit its own slave address; it cannot be master and slave at the same time.

The slave whose address matches that sent by the master pulls I2C_SDA low at the ninth serial clock (D) to return an acknowledge bit.

33.3.3 Data Transfer

When successful slave addressing is achieved, data transfer can proceed (see E in Figure 33-7) on a byte-by-byte basis in the direction specified by the R/W bit sent by the calling master.

Data can be changed only while I2C_SCL is low and must be held stable while I2C_SCL is high, as Figure 33-7 shows. I2C_SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling I2C_SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses. See Figure 33-8.

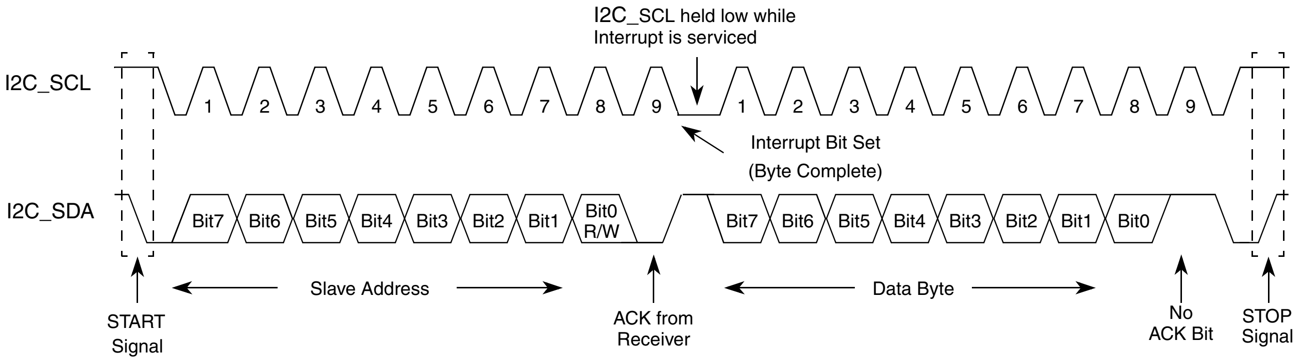


Figure 33-8. Data Transfer

33.3.4 Acknowledge

The transmitter releases the I2C_SDA line high during the acknowledge clock pulse as shown in Figure 33-9. The receiver pulls down the I2C_SDA line during the acknowledge clock pulse so that it remains stable low during the high period of the clock pulse.

If it does not acknowledge the master, the slave receiver must leave I2C_SDA high. The master can then generate a STOP signal to abort data transfer or generate a START signal (repeated start, shown in Figure 33-10 and discussed in Section 33.3.6, "Repeated START") to start a new calling sequence.

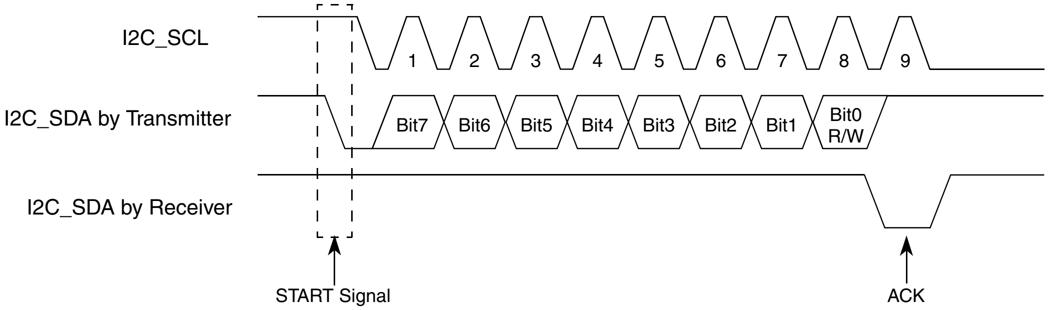


Figure 33-9. Acknowledgement by Receiver

If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end-of-data to the slave. The slave releases I2C_SDA for the master to generate a STOP or START signal (Figure 33-9).

33.3.5 STOP Signal

The master can terminate communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of I2C_SDA while I2C_SCL is at logical high (see F in Figure 33-7). The master can generate a STOP even if the slave has generated an acknowledgment, at which point the slave must release the bus. The master may also generate a START signal following a calling address, without first generating a STOP signal. Refer to Section 33.3.6, "Repeated START."

33.3.6 Repeated START

A repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication, as shown in Figure 33-10. The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.

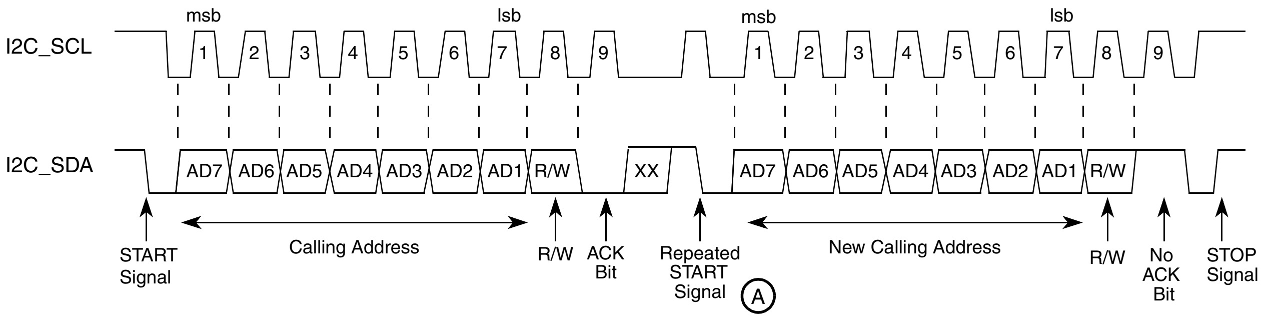


Figure 33-10. Repeated START

Various combinations of read/write formats are then possible:

- The first example in Figure 33-11 is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.
- The second example in Figure 33-11 is the master reading the slave immediately after the first byte. At the moment of the first acknowledge, the master-transmitter becomes a master-receiver and the slave-receiver becomes slave-transmitter.
- In the third example in Figure 33-11, START condition and slave address are repeated using the repeated START signal. This is to communicate with same slave in a different mode without releasing the bus. The master transmits data to the slave first, and then the master reads data from slave by reversing the R/$\overline{W}$ bit.

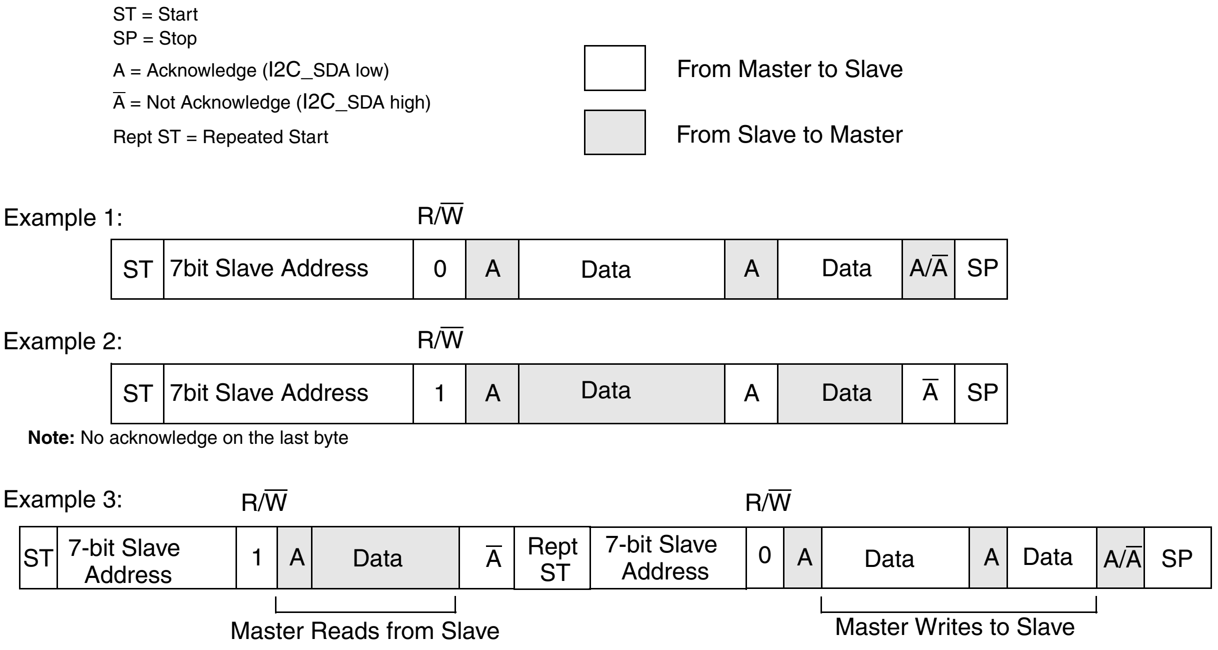


Figure 33-11. Data Transfer, Combined Format

33.3.7 Clock Synchronization and Arbitration

I^{2}C is a true multi-master bus that allows more than one master connected to it. If two or more master devices simultaneously request control of the bus, a clock synchronization procedure determines the bus clock. Because wire-AND logic is performed on the I2C_SCL line, a high-to-low transition on the I2C_SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the I2C_SCL line low until the clock high state is reached. However, change of low to high in this device's clock may not change the state of the I2C_SCL line if another device clock remains within its low period. Therefore, synchronized clock I2C_SCL is held low by the device with the longest low period.

Devices with shorter low periods enter a high wait state during this time (see Figure 33-12). When all devices concerned have counted off their low period, the synchronized clock (I2C_SCL) line is released and pulled high. At this point, the device clocks and the I2C_SCL line are synchronized, and the devices start counting their high periods. The first device to complete its high period pulls the I2C_SCL line low again.

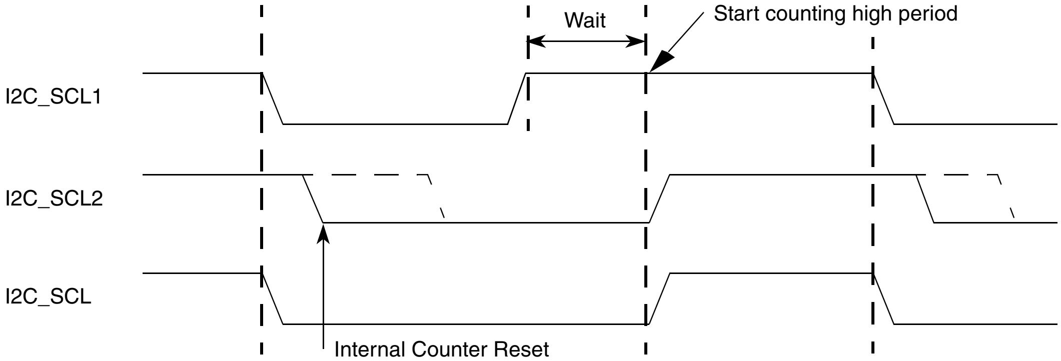


Figure 33-12. Clock Synchronization

A data arbitration procedure determines the relative priority of the contending masters. A bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving I2C_SDA output (see Figure 33-13). In this case, transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I2SR[IAL] to indicate loss of arbitration.

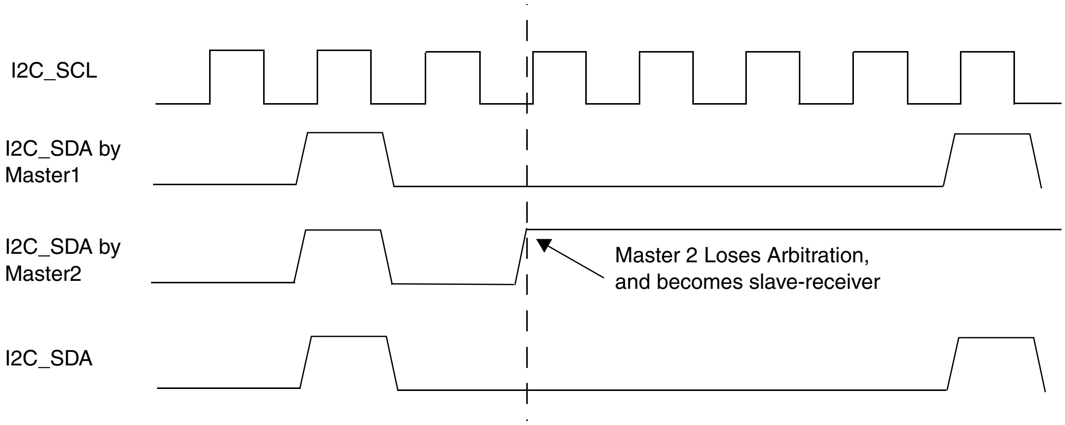


Figure 33-13. Arbitration Procedure

33.3.8 Handshaking and Clock Stretching

The clock synchronization mechanism can acts as a handshake in data transfers. Slave devices can hold I2C_SCL low after completing one byte transfer. In such a case, the clock mechanism halts the bus clock and forces the master clock into wait states until the slave releases I2C_SCL.

Slaves may also slow down the transfer bit rate. After the master has driven I2C_SCL low, the slave can drive I2C_SCL low for the required period and then release it. If the slave I2C_SCL low period is longer than the master I2C_SCL low period, the resulting I2C_SCL bus signal low period is stretched.

33.4 Initialization/Application Information

The following examples show programming for initialization, signaling START, post-transfer software response, signaling STOP, and generating a repeated START.

33.4.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized:

1. Set I2FDR[IC] to obtain I2C_SCL frequency from the system bus clock. See Section 33.2.2, “I2C Frequency Divider Register (I2FDR).”
2. Update the I2ADR to define its slave address.
3. Set I2CR[IEN] to enable the I^{2}C bus interface system.
4. Modify the I2CR to select or deselect master/slave mode, transmit/receive mode, and interrupt-enable or not.

NOTE

If I2SR[IBB] is set when the I^{2}C bus module is enabled, execute the following pseudocode sequence before proceeding with normal initialization code. This issues a STOP command to the slave device, placing it in idle state as if it were power-cycled on.

```
I2CR = 0x0
I2CR = 0xA0
dummy read of I2DR
I2SR = 0x0
I2CR = 0x0
I2CR = 0x80       ; re-enable
```

33.4.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. On a multiple-master bus system, I2SR[IBB] must be tested to determine whether the serial bus is free. If the bus is free (IBB is cleared), the START signal and the first byte (the slave address) can be sent. The data written to the data register comprises the address of the desired slave and the lsb indicates the transfer direction.

The free time between a STOP and the next START condition is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the I2C_SCL period, the processor may need to wait until the I2C is busy after writing the calling address to the I2DR before proceeding with the following instructions.

The following example signals START and transmits the first byte of data (slave address):

1. Check I2SR[IBB]. If it is set, wait until it is clear.
2. After cleared, set to transmit mode by setting I2CR[MTX].
3. Set master mode by setting I2CR[MSTA]. This generates a START condition.
4. Transmit the calling address via the I2DR.
5. Check I2SR[IBB]. If it is clear, wait until it is set and go to step #1.

33.4.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR[ICF], which indicates one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR[IIEN]. Software must first clear I2SR[IIF] in the interrupt routine. Reading from I2DR in receive mode or writing to I2DR in transmit mode can clear I2SR[ICF].

Software can service the I^{2}C I/O in the main program by monitoring the IIF bit if the interrupt function is disabled. Polling should monitor IIF rather than ICF, because that operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode; the address is sent. If master receive mode is required, I2CR[MTX] should be toggled.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

The following is an example of a software response by a master transmitter in the interrupt routine (see Figure 33-14).

1. Clear the I2CR[IIF] flag.
2. Check if acknowledge has been received, I2SR[RXAK].
3. If no ACK, end transmission. Else, transmit next byte of data via I2DR.

33.4.4 Generation of STOP

A data transfer ends when the master signals a STOP, which can occur after all data is sent, as in the following example.

1. Check if acknowledge has been received, I2SR[RXAK]. If no ACK, end transmission and go to step #5.
2. Get value from transmitting counter, TXCNT. If no more data, go to step #5.
3. Transmit next byte of data via I2DR.
4. Decrement TXCNT and go to step #1
5. Generate a stop condition by clearing I2CR[MSTA].

For a master receiver to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last data byte. This is done by setting I2CR[TXAK] before reading the next-to-last byte. Before the last byte is read, a STOP signal must be generated, as in the following example.

1. Decrement RXCNT.
2. If last byte (RXCNT = 0) go to step #4.
3. If next to last byte (RXCNT = 1), set I2CR[TXAK] to disable ACK and go to step #5.
4. This is last byte, so clear I2CR[MSTA] to generate a STOP signal.
5. Read data from I2DR.
6. If there is more data to be read (RXCNT ≠ 0), go to step #1 if desired.

33.4.5 Generation of Repeated START

If the master wants the bus after the data transfer, it can signal another START followed by another slave address without signaling a STOP, as in the following example.

1. Generate a repeated START by setting I2CR[RSTA].
2. Transmit the calling address via I2DR.

33.4.6 Slave Mode

In the slave interrupt service routine, software must poll the I2SR[IAAS] bit to determine if the controller has received its slave address. If IAAS is set, software must set the transmit/receive mode select bit (I2CR[MTX]) according to the I2SR[SRW]. Writing to I2CR clears IAAS automatically. The only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred; interrupts resulting from subsequent data transfers have IAAS cleared. A data transfer can now be initiated by writing information to I2DR for slave transmits, or read from I2DR in slave-receive mode. A dummy read of I2DR in slave/receive mode releases I2C_SCL, allowing the master to send data.

In the slave transmitter routine, I2SR[RXAK] must be tested before sending the next byte of data. Setting RXAK means an end-of-data signal from the master receiver, after which software must switch it from transmitter to receiver mode. Reading I2DR releases I2C_SCL so the master can generate a STOP signal.

33.4.7 Arbitration Lost

If several devices try to engage the bus at the same time, one becomes master. Hardware immediately switches devices that lose arbitration to slave receive mode. Data output to I2C_SDA stops, but I2C_SCL continues generating until the end of the byte during which arbitration is lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with I2SR[IAL] set and I2CR[MSTA] cleared.

If a non-master device tries to transmit or execute a START, hardware inhibits the transmission, clears MSTA without signaling a STOP, generates an interrupt to the CPU, and sets IAL to indicate a failed attempt to engage the bus. When considering these cases, slave service routine should first test IAL and software should clear it if it is set.

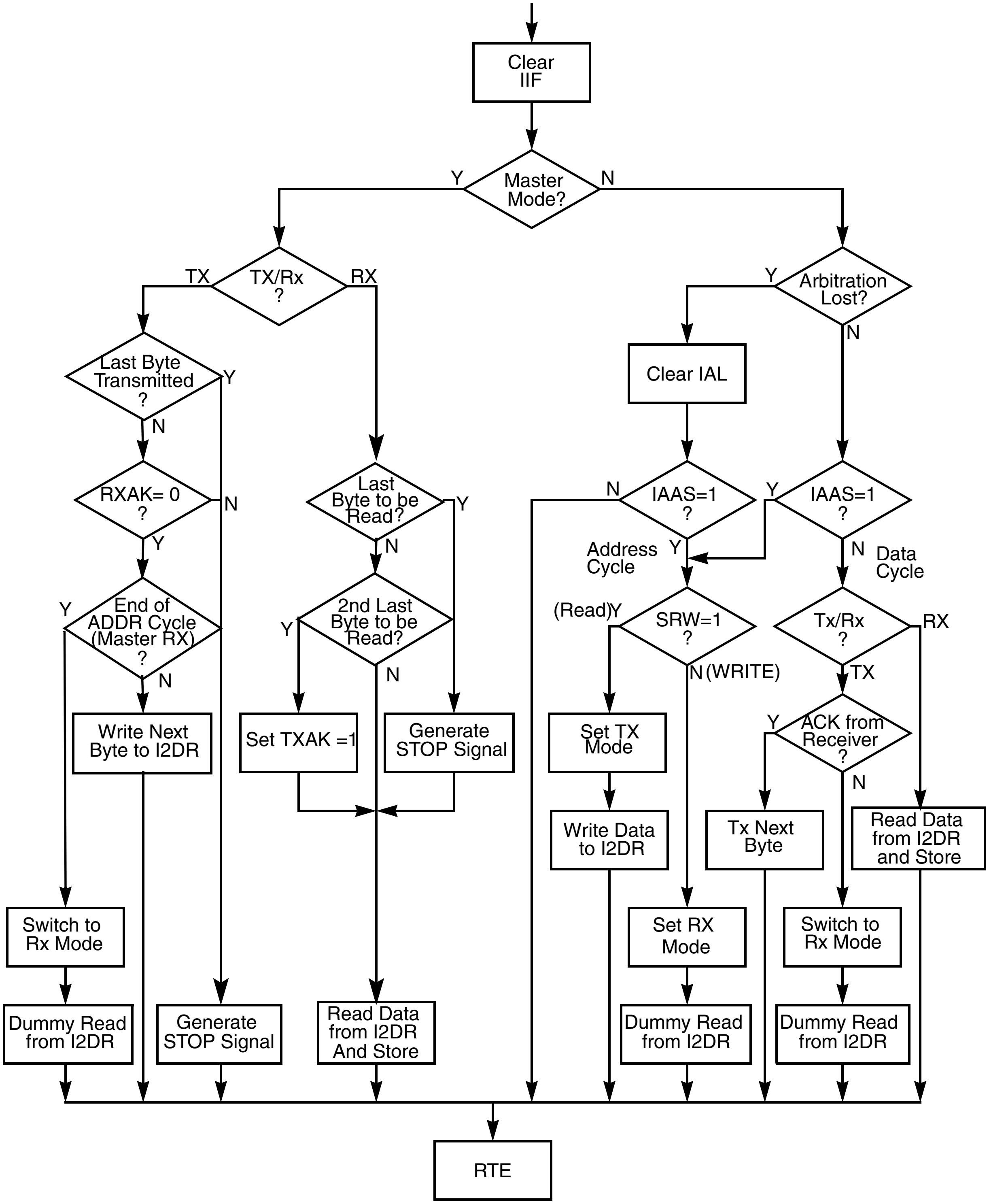


Figure 33-14. Flow-Chart of Typical I²C Interrupt Routine

Chapter 34
Debug Module

34.1 Introduction

This chapter describes the revision D+ enhanced hardware debug module.

34.1.1 Block Diagram

The debug module is shown in Figure 34-1.

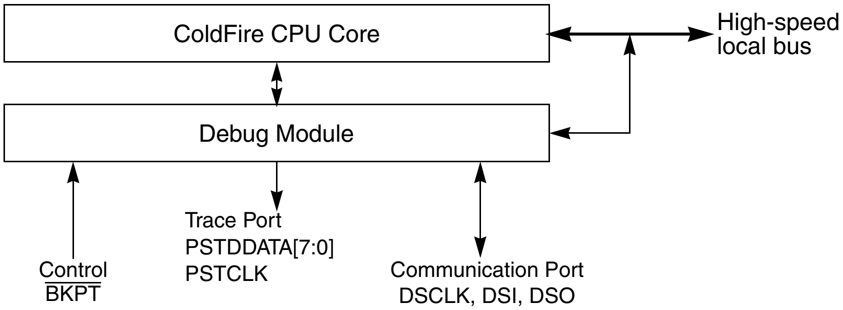


Figure 34-1. Processor/Debug Module Interface

34.1.2 Overview

Debug support is divided into three areas:

- Real-time trace support—The ability to determine the dynamic execution path through an application is fundamental for debugging. The ColdFire solution implements an 8-bit parallel output bus that reports processor execution status and data to an external emulator system. See Section 34.4.4, "Real-Time Trace Support".
- Background debug mode (BDM)—Provides low-level debugging in the ColdFire processor complex. In BDM, the processor complex is halted and a variety of commands can be sent to the processor to access memory, registers, and peripherals. The external emulator uses a three-pin, serial, full-duplex channel. See Section 34.4.1, "Background Debug Mode (BDM)," and Section 34.3, "Memory Map/Register Definition".
- Real-time debug support—BDM requires the processor to be halted, which many real-time embedded applications cannot do. Debug interrupts let real-time systems execute a unique service routine that can quickly save the contents of key registers and variables and return the system to normal operation. External development systems can access saved data, because the hardware supports concurrent operation of the processor and BDM-initiated commands. In addition, the option allows interrupts to occur. See Section 34.4.2, "Real-Time Debug Support".

The first version 2 ColdFire core devices implemented the original debug architecture, now called revision A. Based on feedback from customers and third-party developers, enhancements have been added to succeeding generations of ColdFire cores. For revision A, CSR[HRL] is 0. See Section 34.3.2, “Configuration/Status Register (CSR)”.

Revision B (and B+) of the debug architecture offers more flexibility for configuring the hardware breakpoint trigger registers and removing the restrictions involving concurrent BDM processing while hardware breakpoint registers are active. Revision B+ adds three additional PC breakpoint registers. For revision B, CSR[HRL] is 1, and for revision B+, CSR[HRL] is 0x9.

Revision C of the debug architecture more than doubles the on-chip breakpoint registers and provides an ability to interrupt debug service routines. This revision also combines the PST and DDATA signals into PSTDDATA[7:0]. Because real-time trace information appears as a sequence of 4-bit values, there are no alignment restrictions. In other words, PST values and operands may appear on either nibble of PSTDDATA. For revision C, CSR[HRL] is 2.

The addition of the memory management unit (MMU) to the baseline architecture requires corresponding enhancements to the ColdFire debug functionality, resulting in revision D. For revision D, the revision level bit, CSR[HRL], is 3.

With software support, the MMU can provide a demand-paged, virtual address environment. To support debugging in this virtual environment, the debug enhancements are primarily related to the expansion of the virtual address to include the 8-bit address space identifier (ASID). Conceptually, the virtual address is expanded to a 40-bit value: the 8-bit ASID plus the 32-bit address.

The expansion of the virtual address affects two major debug functions:

- The ASID is optionally included in the specification of the hardware breakpoint registers. As an example, the four PC breakpoint registers are each expanded by 8 bits, so that a specific ASID value may be programmed as part of the breakpoint instruction address. Likewise, each operand address/data breakpoint register is expanded to include an ASID value. Finally, control registers define if and how the ASID is to be included in the breakpoint comparison trigger logic.
- The debug module implements the concept of ownership trace in which the ASID value may be optionally displayed as part of the real-time trace functionality. When enabled, real-time trace displays instruction addresses on every change-of-flow instruction that is not absolute or PC-relative. For Rev. D, this instruction address display optionally includes the contents of the ASID, thus providing the complete instruction virtual address on these instructions.
- Additionally when a SYNC_PC serial BDM command is loaded from the external development system, the processor optionally displays the complete virtual instruction address, including the 8-bit ASID value.

In addition to these ASID-related changes, the MMU control registers are accessible by using serial BDM commands. The same BDM access capabilities are also provided for the EMAC programming model.

Finally, a serial BDM command is implemented (FORCE_TA) to assist debugging when a software error generates an incorrect memory address that hangs the external bus. The BDM command attempts to break this condition by forcing a bus termination.

Revision D+ adds the ignore pending interrupt bit (CSR[IPI]) for debug revision D. (This bit is included in revision A, B, and B+). For revision D+, the revision level (CSR[HRL]) is 0xB.

The following table summarizes the various debug revisions.

Table 34-1. Debug Revision Summary

Revision	CSR[HRL]		Enhancements
A	0000	—	Initial debug revision
B	0001	—	BDM command execution does not affect hardware breakpoint logic Added BDM address attribute register (BAAR) $\overline{\text{BKPT}}$ configurable interrupt (CSR[BKD]) Level 1 and level 2 triggers on OR condition, in addition to AND SYNC_PC command to display the processor's current PC
B+	1001	—	3 additional PC breakpoint registers PBR1–3
C	0010	—	Combined PST and DDATA signals Adds breakpoint registers Supports normal interrupt request service during debug Redefinition of the PST values for the RTS instruction
D	0011	—	MMU enhancements to support ASID FORCE_TA command
D+	1011	—	Added CSR[IPI] for revision D

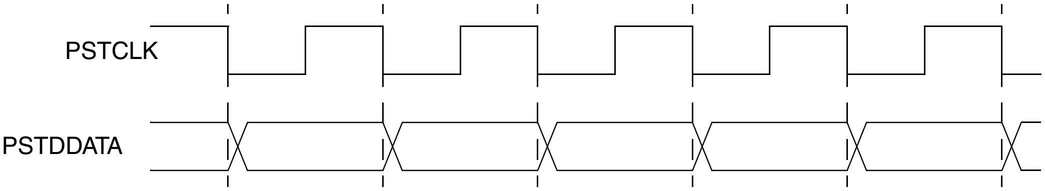
34.2 Signal Descriptions

Table 34-2 describes debug module signals. All ColdFire debug signals are unidirectional and related to a rising edge of the processor core's clock signal. The standard 26-pin debug connector is shown in Section 34.4.6, "Freescale-Recommended BDM Pinout".

Table 34-2. Debug Module Signals

Signal	Description
Development Serial Clock (DSCLK)	Internally synchronized input. (The logic level on DSCLK is validated if it has the same value on two consecutive rising bus clock edges.) Clocks the serial communication port to the debug module during packet transfers. Maximum frequency is 1/5 the processor status clock (PSTCLK). At the synchronized rising edge of DSCLK, the data input on DSI is sampled and DSO changes state.
Development Serial Input (DSI)	Internally synchronized input that provides data input for the serial communication port to the debug module after the DSCLK has been seen as high (logic 1).
Development Serial Output (DSO)	Provides serial output communication for debug module responses. DSO is registered internally. The output is delayed from the validation of DSCLK high.
Breakpoint ($\overline{\text{BKPT}}$)	Input requests a manual breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status/debug data signals (PSTDDATA[7:0]) as multiple cycles of 0xF. If CSR[BKD] is set (disabling normal $\overline{\text{BKPT}}$ functionality), asserting $\overline{\text{BKPT}}$ generates a debug interrupt exception in the processor.

Table 34-2. Debug Module Signals (continued)

Signal	Description
Processor Status Clock (PSTCLK)	Half-speed version of the processor clock. Its rising edge appears in the center of the two-processor-cycle window of valid PSTDDATA output. PSTCLK indicates when the development system should sample PSTDDATA values. The following figure shows PSTCLK timing with respect to PSTDDATA.  If real-time trace is not used, setting CSR[PCD] keeps PSTCLK and PSTDDATA outputs from toggling without disabling triggers. Non-quiescent operation can be reenabled by clearing CSR[PCD], although the external development systems must resynchronize with the PSTDDATA output. PSTCLK starts clocking only when the first non-zero PST value (0xC, 0xD, or 0xF) occurs during system reset exception processing. Table 34-30 describes PST values.
Processor Status/Debug Data (PSTDDATA[7:0])	These outputs, which change on the negative edge of PSTCLK, indicate processor status and captured address and data values and are discussed more thoroughly in Section 34.2.1, “Processor Status/Debug Data (PSTDDATA[7:0]).”

34.2.1 Processor Status/Debug Data (PSTDDATA[7:0])

Processor status data outputs indicate processor status and captured address and data values. They operate at half the processor’s frequency. Given that real-time trace information appears as a sequence of 4-bit data values, there are no alignment restrictions; that is, the processor status (PST) values and operands may appear on either nibble of PSTDDATA[7:0]. The upper nibble (PSTDDATA[7:4]) is the more significant and yields values first.

The CSR register controls capturing of data values to be presented on PSTDDATA. Executing the WDDATA instruction captures data that is displayed on PSTDDATA too. These signals are updated each processor cycle and display two values at a time for two processor clock cycles. Table 34-3 shows the PSTDDATA output for the processor’s sequential execution of single-cycle instructions (A, B, C, D...). Cycle counts are shown relative to processor frequency. These outputs indicate the current processor pipeline status and are not related to the current bus transfer.

Table 34-3. PSTDDATA: Sequential Execution of Single-Cycle Instructions

Cycles	PSTDDATA[7:0]
T+0, T+1	{PST for A, PST for B}
T+2, T+3	{PST for C, PST for D}
T+4, T+5	{PST for E, PST for F}

The signal timing for the example in Table 34-3 is shown in Figure 34-2.

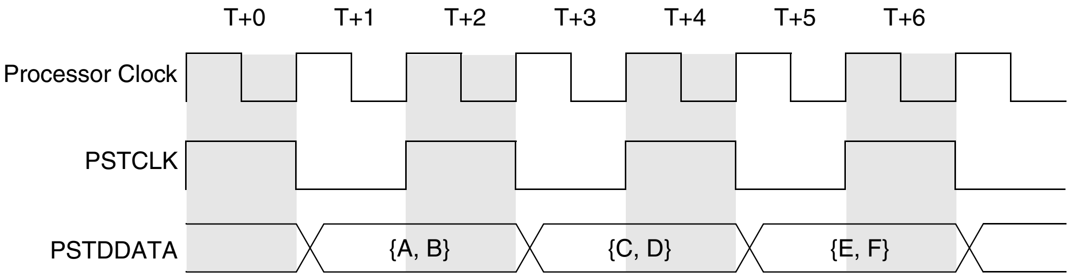


Figure 34-2. PSTDDATA: Single-Cycle Instruction Timing

Table 34-4 shows the case where a PSTDDATA module captures a memory operand on a simple load instruction: `move.l <mem>,Rx`.

Table 34-4. PSTDDATA: Data Operand Captured

Cycle	PSTDDATA[7:0]
T	{PST for move.l, PST marker for captured operand) = {0x1, 0xB}
T+1	{0x1, 0xB}
T+2	{Operand[3:0], Operand[7:4]}
T+3	{Operand[3:0], Operand[7:4]}
T+4	{Operand[11:8], Operand[15:12]}
T+5	{Operand[11:8], Operand[15:12]}
T+6	{Operand[19:16], Operand[23:20]}
T+7	{Operand[19:16], Operand[23:20]}
T+8	{Operand[27:24], Operand[31:28]}
T+9	{Operand[27:24], Operand[31:28]}
T+10	(PST for next instruction)
T+11	(PST for next instruction,...)

NOTE

A PST marker and its data display are sent contiguously. Except for this transmission, the IDLE status (0x0) can appear anytime. Again, given that real-time trace information appears as a sequence of 4-bit values, there are no alignment restrictions. That is, PST values and operands may appear on either nibble of PSTDDATA.

34.3 Memory Map/Register Definition

In addition to the existing BDM commands that provide access to the processor’s registers and the memory subsystem, the debug module contain a number of registers to support the required functionality. These registers are also accessible from the processor’s supervisor programming model by executing the WDEBUG instruction (write only). Therefore, the breakpoint hardware in debug module can be read or written by the external development system using the debug serial interface or written by the operating system running on the processor core. Software guarantees that accesses to these resources are serialized and logically consistent. Hardware provides a locking mechanism in CSR to allow external development system to disable any attempted writes by the processor to the breakpoint registers (setting CSR[IPW]). BDM commands must not be issued if the ColdFire processor is using the WDEBUG instruction to access debug module registers, or the resulting behavior is undefined. The DSCLK must be quiescent during operation of the WDEBUG command.

These registers, shown in Table 34-5, are treated as 32-bit quantities, regardless of the number of implemented bits. These registers are also accessed through the BDM port by the commands, WDMREG and RDMREG, described in Section 34.4.1.5, “BDM Command Set”. These commands contain a 5-bit field, DRc, that specifies the register, as shown in Table 34-5.

Table 34-5. Debug Module Memory Map

DRc[4–0]	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W See Note	0x00B0_0000	34.3.2/34-9
0x05	BDM address attribute register (BAAR)	32[1]	W	0x05	34.3.3/34-11
0x06	Address attribute trigger register (AATR)	32	W	0x0000_0005	34.3.4/34-12
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	34.3.5/34-14
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined	34.3.6/34-17
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined	34.3.6/34-17
0x0A	PC breakpoint ASID control (PBAC)	32	W	Undefined	34.3.7/34-18
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined	34.3.8/34-19
0x0D	Address breakpoint low register (ABLR)	32	W	Undefined	34.3.8/34-19
0x0E	Data breakpoint register (DBR)	32	W	Undefined	34.3.9/34-20
0x0F	Data breakpoint mask register (DBMR)	32	W	Undefined	34.3.9/34-20
0x14	PC breakpoint ASID register (PBASID)	32	W	Undefined	34.3.10/34-21
0x16	Address attribute trigger register 1 (AATR1)	32	W	0x0005	34.3.4/34-12
0x17	Extended trigger definition register (XTDR)	32	W	0x0000_0000	34.3.11/34-22
0x18	PC breakpoint register 1 (PBR1)	32	W	See Section	34.3.6/34-17
0x1A	PC breakpoint register 2 (PBR2)	32	W	See Section	34.3.6/34-17
0x1B	PC breakpoint register 3 (PBR3)	32	W	See Section	34.3.6/34-17
0x1C	Address high breakpoint register 1 (ABHR1)	32	W	Undefined	34.3.8/34-19
0x1D	Address low breakpoint register 1 (ABLR1)	32	W	Undefined	34.3.8/34-19
0x1E	Data breakpoint register 1 (DBR1)	32	W	Undefined	34.3.9/34-20
0x1F	Data breakpoint mask register 1 (DBMR1)	32	W	Undefined	34.3.9/34-20

[1] Each debug register is accessed as a 32-bit register; reserved fields are not used (don’t care).

NOTE

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction. These control registers are write-only from the programming model and they can be written through the BDM port using the WDMREG command. In addition, the configuration/status register (CSR) can be read through the BDM port using the RDMREG command.

The ColdFire debug architecture supports a number of hardware breakpoint registers, that can be configured into single- or double-level triggers based on the PC or operand address ranges with an optional inclusion of specific data values. With the addition of the MMU capabilities, the breakpoint specifications must be expanded to optionally include the address space identifier (ASID) in these user-programmable virtual address triggers.

The core includes four PC breakpoint triggers and two sets of operand address breakpoint triggers, each with two independent address registers (to allow specification of a range) and a data breakpoint with masking capabilities. Core breakpoint triggers are accessible through the serial BDM interface or written through the supervisor programming model using the WDEBUG instruction.

Two ASID-related registers (PBAC and PBASID) are added for the PC breakpoint qualification, and two existing registers (AATR and AATR1) are expanded for the address breakpoint qualification.

34.3.1 Shared Debug Resources

The debug module revision A implementation provides a common hardware structure for BDM and breakpoint functionality. Certain hardware structures are used for BDM and breakpoint purposes as shown in Table 34-6.

Table 34-6. Shared BDM/Breakpoint Hardware

Register	BDM Function	Breakpoint Function
AATR	Bus attributes for all memory commands	Attributes for address breakpoint
ABHR	Address for all memory commands	Address for address breakpoint
DBR	Data for all BDM write commands	Data for data breakpoint

Therefore, loading a register to perform a specific function that shares hardware resources is destructive to the shared function. For example, if an operand address breakpoint is loaded into the debug module, a BDM command to access memory overwrites an address breakpoint in ABHR. If a data breakpoint is configured, a BDM write command overwrites the data breakpoint in DBR.

Revision B added hardware registers to eliminate these shared functions. The BAAR is used to specify bus attributes for BDM memory commands and has the same format as the LSB of the AATR. The registers containing the BDM memory address and the BDM data are not program visible.

34.3.2 Configuration/Status Register (CSR)

The CSR defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic. CSR is write-only from the programming model. It can be read from and written to through the BDM port. CSR is accessible in supervisor mode as debug control register 0x00 using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands.

DRc[4:0]: 0x00 (CSR)

Access: Supervisor write-only
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BSTAT				FOF	TRG	HALT	BKPT	HRL				0	BKD	PCD	IPW
W																
Reset	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAP	TRC	EMU	DDC		UHE	BTB		0	NPL	IPI	SSM	OTE	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-3. Configuration/Status Register (CSR)

Table 34-7. CSR Field Descriptions

Field	Description
31–28 BSTAT	Breakpoint Status. Provides read-only status (from the BDM port only) information concerning hardware breakpoints. Also output on PSTDDATA when it is not displaying PST or other processor data. BSTAT is cleared by a TDRor XTDR write or by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. 0000 No breakpoints enabled 0001 Waiting for level-1 breakpoint 0010 Level-1 breakpoint triggered 0101 Waiting for level-2 breakpoint 0110 Level-2 breakpoint triggered Else Reserved
27 FOF	Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM.
26 TRG	Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset or the debug GO command clear TRG.
25 HALT	Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset or the debug GO command clear HALT.
24 BKPT	Breakpoint assert. If BKPT is set, $\overline{\text{BKPT}}$ was asserted, forcing the processor into BDM. Reset or the debug GO command clear BKPT.

Table 34-7. CSR Field Descriptions (continued)

Field	Description
23–20 HRL	Hardware revision level. Indicates, from the BDM port only, the level of debug module functionality. An emulator could use this information to identify the level of functionality supported. 0000 Revision A 0001 Revision B 0010 Revision C 0011 Revision D 1001 Revision B+ 1011 Revision D+ (This is the value used for this device) 1111 Revision D+PSTB
19	Reserved, must be cleared.
18 BKD	Breakpoint disable. Disables the normal $\overline{\text{BKPT}}$ input signal functionality, and allows the assertion of this pin to generate a debug interrupt. 0 Normal operation 1 $\overline{\text{BKPT}}$ is edge-sensitive: a high-to-low edge on $\overline{\text{BKPT}}$ signals a debug interrupt to the ColdFire core. The processor makes this interrupt request pending until the next sample point occurs, when the exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for nesting debug interrupts.
17 PCD	PSTCLK disable. 0 PSTCLK is fully operational 1 Disables the generation of the PSTCLK and PSTDDATA output signals, and forces these signals to remain quiescent
16 IPW	Inhibit processor writes. Setting IPW inhibits processor-initiated writes to the debug module's programming model registers. Only commands from the external development system can modify IPW.
15 MAP	Force processor references in emulator mode. 0 All emulator-mode references are mapped into supervisor code and data spaces. 1 The processor maps all references while in emulator mode to a special address space, TT equals 10, TM equals 101 or 110. The internal SRAM and caches are disabled.
14 TRC	Force emulation mode on trace exception. 0 The processor enters supervisor mode 1 The processor enters emulator mode when a trace exception occurs
13 EMU	Force emulation mode. 0 Do not force emulator mode 1 The processor begins executing in emulator mode. See Section 34.4.2.2, "Emulator Mode".
12–11 DDC	Debug data control. Controls operand data capture for PSTDDATA, which displays the number of bytes defined by the operand reference size before the actual data; byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple PSTCLK clock cycles). See Table 34-30. 00 No operand data is displayed. 01 Capture all write data. 10 Capture all read data. 11 Capture all read and write data.
10 UHE	User halt enable. Selects the CPU privilege level required to execute the HALT instruction. 0 HALT is a supervisor-only instruction. 1 HALT is a supervisor/user instruction.

Table 34-7. CSR Field Descriptions (continued)

Field	Description
9–8 BTB	Branch target bytes. Defines the number of bytes of branch target address PSTDDATA displays. 00 0 bytes 01 Lower 2 bytes of the target address 10 Lower 3 bytes of the target address 11 Entire 4-byte target address See Section 34.4.4.1, “Begin Execution of Taken Branch (PST = 0x5)”.
7	Reserved, must be cleared.
6 NPL	Non-pipelined mode. Determines whether the core operates in pipelined mode or not. 0 Pipelined mode 1 Non-pipelined mode. The processor effectively executes one instruction at a time with no overlap. This adds at least 5 cycles to the execution time of each instruction. Superscalar instruction dispatch is disabled when operating in this mode. Given an average execution latency of 1.6 cycles/instruction, throughput in non-pipeline mode would be 6.6 cycles/instruction, approximately 25% or less of pipelined performance. Regardless of the NPL state, a triggered PC breakpoint is always reported before the triggering instruction executes. In normal pipeline operation, occurrence of an address and/or data breakpoint trigger is imprecise. In non-pipeline mode, triggers are always reported before the next instruction begins execution and trigger reporting can be considered precise. An address or data breakpoint should always occur before the next instruction begins execution. Therefore, the occurrence of the address/data breakpoints should be guaranteed.
5 IPI	Ignore pending interrupts. 0 Core services any pending interrupt requests that were signalled while in single-step mode. 1 Core ignores any pending interrupt requests signalled while in single-instruction-step mode.
4 SSM	Single-Step Mode. Setting SSM puts the processor in single-step mode. 0 Normal mode. 1 Single-step mode. The processor halts after execution of each instruction. While halted, any BDM command can be executed. On receipt of the GO command, the processor executes the next instruction and halts again. This process continues until SSM is cleared.
3 OTE	Ownership-trace enable. Enables the display of the ASID on the PSTDDATA outputs upon entrance into user mode, a load of the ASID by a MOVEC instruction, or the execution of a BDM SYNC_PC command. 0 No ASID displayed 1 ASID displayed on PSTDDATA outputs
2	Reserved, must be cleared.
1 FDBG	Force the debug mode core output signal (to the on-chip peripherals). The debug mode output is logically defined as: Debug mode output = CSR[FDBG] | ($\overline{\text{CSR[DBGH]}}$ and Core is halted) 0 Debug mode output is not forced asserted. 1 Debug mode output core output signal is asserted.
0 DBGH	Disable debug signal assertion during core halt. The debug mode output (to the on-chip peripherals) is logically defined as: Debug mode output = CSR[FDBG] | ($\overline{\text{CSR[DBGH]}}$ and Core is halted) 0 Debug mode output is asserted when the core is halted. 1 Debug mode output is not asserted when the core is halted.

34.3.3 BDM Address Attribute Register (BAAR)

The BAAR register defines the address space for memory-referencing BDM commands. BAAR[R, SZ] are loaded directly from the BDM command, while the low-order 5 bits can be programmed from the external development system. To maintain compatibility with revision A, BAAR is loaded any time the AATR is written. The BAAR is initialized to a value of 0x05, setting supervisor data as the default address space.

DRc[4:0]: 0x05 (BAAR)

Access: Supervisor write-only
BDM write-only

	7	6	5	4	3	2	1	0
R								
W	R	SZ		TT		TM		
Reset:	0	0	0	0	0	1	0	1

Figure 34-4. BDM Address Attribute Register (BAAR)

Table 34-8. BAAR Field Descriptions

Field	Description
7 R	Read/Write. 0 Write 1 Read
6–5 SZ	Size. 00 Longword 01 Byte 10 Word 11 Reserved
4–3 TT	Transfer Type. See the TT definition in the AATR description, Section 34.3.4, “Address Attribute Trigger Registers (AATR, AATR1)”.
2–0 TM	Transfer Modifier. See the TM definition in the AATR description, Section 34.3.4, “Address Attribute Trigger Registers (AATR, AATR1)”.

34.3.4 Address Attribute Trigger Registers (AATR, AATR1)

The AATR and AATR1 define address attributes and a mask to be matched in the trigger. The register value is compared with address attribute signals from the processor’s local high-speed bus, as defined by the setting of the trigger definition register (TDR) for AATR and the extended trigger definition register (XTDR) for AATR1. AATR*n* is accessible in supervisor mode as debug control register 0x06 using the WDEBUG instruction and through the BDM port using the WDMREG command.

This register is expanded to include an optional ASID specification and a control bit that enables the use of the ASID field.

DRc[4:0]: 0x06 (AATR)
0x16 (AATR1)

Access: Supervisor write-only
BDM write-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	0	0	0	0	0	0	0	ASID CTRL	AATRASID							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	RM	SZM		TTM		TMM			R	SZ		TT		TM		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figure 34-5. Address Attribute Trigger Registers (AATR, AATR1)

Table 34-9. AATR*n* Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 ASIDCTRL	ABLR/ABHR/AATR address breakpoint ASID enable. Corresponds to the ASID control enable for the address breakpoint defined in ABLR, ABHR, and AATR. 0 Disable ASID qualifier (reset default) 1 Enable ASID qualifier
23–16 AATRASID	ABLR/ABHR/AATR ASID. Corresponds to the ASID to be included in the address breakpoint specified by ABLR, ABHR, and AATR.
15 RM	Read/write Mask. Setting RM masks R in address comparisons.
14–13 SZM	Size Mask. Setting an SZM bit masks the corresponding SZ bit in address comparisons.
12–11 TTM	Transfer Type Mask. Setting a TTM bit masks the corresponding TT bit in address comparisons.
10–8 TMM	Transfer Modifier Mask. Setting a TMM bit masks the corresponding TM bit in address comparisons.
7 R	Read/Write. R is compared with the $R/\overline{W}$ signal of the processor’s local bus.
6–5 SZ	Size. Compared to the processor’s local bus size signals. 00 Longword 01 Byte 10 Word 11 Reserved

Table 34-9. AATR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>4–3
TT</td><td>Transfer Type. Compared with the local bus transfer type signals.
00 Normal processor access
01 Reserved
10 Emulator mode access
11
These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding indicates an external or DMA access (for backward compatibility). These bits affect the TM bits.</td></tr>
<tr><td>2–0
TM</td><td>Transfer Modifier. Compared with the local bus transfer modifier signals, which give supplemental information for each transfer type. These bits also define the TM encoding for BDM memory commands (for backward compatibility).
<table>
<tr><th>TM</th><th>TT=00
(normal mode)</th><th>TT=10
(emulator mode)</th></tr>
<tr><td>000</td><td>Data and instruction cache line push</td><td>Reserved</td></tr>
<tr><td>001</td><td>User data access</td><td>Reserved</td></tr>
<tr><td>010</td><td>User code access</td><td>Reserved</td></tr>
<tr><td>011</td><td>Instruction cache invalidate</td><td>Reserved</td></tr>
<tr><td>100</td><td>Data cache push/
Instruction cache invalidate</td><td>Reserved</td></tr>
<tr><td>101</td><td>Supervisor data access</td><td>Emulator mode access</td></tr>
<tr><td>110</td><td>Supervisor code access</td><td>Emulator code access</td></tr>
<tr><td>111</td><td>INTOUCH instruction access</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

34.3.5 Trigger Definition Register (TDR)

The TDR configures the operation of the hardware breakpoint logic corresponding with the ABHR/ABLR/AATR, PBR/PBR1/PBR2/PBR3/PBMR, and DBR/DBMR registers within the debug module. TDR controls the actions taken under the defined conditions with the XTDR and its associated debug registers. Breakpoint logic may be configured as one- or two-level triggers. TDR[31–16] or XTDR[31–16] bits define second-level triggers, and bits 15–0 define first-level triggers.

NOTE

The debug module has no hardware interlocks to prevent spurious breakpoint triggers while the breakpoint registers are being loaded. Disable TDR and XTDR (by clearing TDR[29,13] and XTDR[29,13]) before defining triggers.

When performing a level-1, level-2, and level-1 breakpoint sequence, TDR[29] must be cleared in the level-2 breakpoint handler for the second level-1 breakpoint to occur.

A write to TDR clears the CSR trigger status bits, CSR[BSTAT]. TDR is accessible in supervisor mode as debug control register 0x07 using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x07 (TDR)

Access: Supervisor write-only
BDM write-only

			Second Level Trigger													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	TRC		L2EBL	L2ED							L2DI	L2EA			L2EPC	L2PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

			First Level Trigger													
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	0	0	L1EBL	L1ED							L1DI	L1EA			L1EPC	L1PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-6. Trigger Definition Register (TDR)

Table 34-10. TDR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–30
TRC</td><td>Trigger Response Control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on PSTDDATA.
00 Display on PSTDDATA only
01 Processor halt
10 Debug interrupt
11 Reserved</td></tr>
<tr><td>29
L2EBL</td><td>Enable Level 2 Breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 2 breakpoints
1 Enables all level 2 breakpoint triggers</td></tr>
<tr><td>28–22
L2ED</td><td>Enable Level 2 Data Breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>28</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>27</td><td>Lower data word.</td></tr>
<tr><td>26</td><td>Upper data word.</td></tr>
<tr><td>25</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>24</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>23</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>22</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
</table>

Table 34-10. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>21
L2DI</td><td>Level 2 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
<tr><td>20–18
L2EA</td><td>Enable Level 2 Address Breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>17
L2EPC</td><td>Enable Level 2 PC Breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint where the trigger is defined by the logical summation of:
$$(\mathrm{PBR0\ and\ }\overline{\mathrm{PBMR}})\ |\ \mathrm{PBR1}\ |\ \mathrm{PBR2}\ |\ \mathrm{PBR3} \qquad \textit{Eqn. 34-1}$$</td></tr>
<tr><td>16
L2PCI</td><td>Level 2 PC Breakpoint Invert.
0 The PC breakpoint is defined within the region defined by PBR<i>n</i> and PBMR.
1 The PC breakpoint is defined outside the region defined by PBR<i>n</i> and PBMR.</td></tr>
<tr><td>15–14</td><td>Reserved, must be cleared.</td></tr>
<tr><td>13
L1EBL</td><td>Enable Level 1 Breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 1 breakpoints
1 Enables all level 1 breakpoint triggers</td></tr>
<tr><td>12–6
L1ED</td><td>Enable Level 1 Data Breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all L1ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>12</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>11</td><td>Lower data word.</td></tr>
<tr><td>10</td><td>Upper data word.</td></tr>
<tr><td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
</table>

Table 34-10. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
L1DI</td><td>Level 1 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
<tr><td>4–2
L1EA</td><td>Enable Level 1 Address Breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>1
L1EPC</td><td>Enable Level 1 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>0
L1PCI</td><td>Level 1 PC Breakpoint Invert.
0 The PC breakpoint is defined within the region defined by PBR<i>n</i> and PBMR.
1 The PC breakpoint is defined outside the region defined by PBR<i>n</i> and PBMR.</td></tr>
</table>

34.3.6 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

The PBR*n* registers define an instruction address for use as part of the trigger. These registers' contents are compared with the processor's program counter register when the appropriate valid bit is set (for PBR1–3) and TDR or XTDR are configured appropriately. PBR0 bits are masked by setting corresponding PBMR bits (PBMR has no effect on PBR1–3). Results are compared with the processor's program counter register, as defined in TDR or XTDR. Breakpoint registers, PBR1–3, have no masking associated with them. The contents of the breakpoint registers are compared with the processor's program counter register when TDR is configured appropriately.

The PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command using values shown in Section 34.4.1.5, "BDM Command Set".

DRc[4:0]: 0x08 (PBR0)

Access: Supervisor write-only
BDM 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
R																																
W	Address																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 34-7. PC Breakpoint Register (PBR0)

Table 34-11. PBR0 Field Descriptions

Field	Description
31–0 Address	PC Breakpoint Address. The address to be compared with the PC as a breakpoint trigger. **Note:** PBR0[0] should always be loaded with a 0.

DRc[4:0]: 0x18 (PBR1)
0x1A (PBR2)
0x1B (PBR3)

Access: Supervisor write-only
BDM 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
R																																
W	Address																															V
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0	0

Figure 34-8. PC Breakpoint Register *n* (PBR*n*)

Table 34-12. PBR*n* Field Descriptions

Field	Description
31–1 Address	PC Breakpoint Address. The 31-bit address to be compared with the PC as a breakpoint trigger.
0 V	Valid Bit. This bit must be set for the PC breakpoint to occur at the address specified in the Address field. 0 PBR is disabled. 1 PBR is enabled.

Figure 34-9 shows PBMR. PBMR is accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command. PBMR only masks PBR0.

DRc[4:0]: 0x09 (PBMR)

Access: Supervisor write-only
BDM 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
R																																
W	Mask																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 34-9. PC Breakpoint Mask Register (PBMR)

Table 34-13. PBMR Field Descriptions

Field	Description
31–0 Mask	PC Breakpoint Mask. 0 The corresponding PBR0 bit is compared to the appropriate PC bit. 1 The corresponding PBR0 bit is ignored.

34.3.7 PC Breakpoint ASID Control Register (PBAC)

The PBAC register configures the breakpoint qualification for each PC breakpoint register (PBR*n*). Four bits are dedicated for each breakpoint register and specify how the ASID is used in PC breakpoint qualification.

The four-bit field correspond directly to the PBRn registers and are functionally identical. They enable or disable ASID, supervisor mode, and user mode breakpoint qualification. Reset clears these fields, disabling qualifications, and defaulting to the revision C debug module functionality.

DRc[4:0]: 0x0A (PBAC)

Access: Supervisor read/write
BDM 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
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PBR3AC				PBR2AC				PBR1AC				PBR0AC			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-10. PC Breakpoint ASID Control Register (PBAC)

Table 34-14. PBAC Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–16</td><td>Reserved, must be cleared.</td></tr>
<tr><td>15–12
PBR3AC
11–8
PBR2AC
7–4
PBR1AC
3–0
PBR0AC</td><td>PBR<i>n</i> ASID control. Corresponds to the ASID control associated with PBR<i>n</i>. Determines whether the ASID is included in the PC breakpoint comparison and whether the operating mode (supervisor or user) is included in the comparison logic.
<table>
<tr><th>PBR<i>n</i>AC[3]
Not Used</th><th>PBR<i>n</i>AC[2]
ASID
Included</th><th>PBR<i>n</i>AC[1]
Mode
Qualification</th><th>PBR<i>n</i>AC[0]
User or
Supervisor</th><th>Description</th></tr>
<tr><td>x</td><td>0</td><td>0</td><td>x</td><td>No ASID nor mode qualification</td></tr>
<tr><td>x</td><td>0</td><td>1</td><td>0</td><td>No ASID qualification; user mode qualification enabled</td></tr>
<tr><td>x</td><td>0</td><td>1</td><td>1</td><td>No ASID qualification; supervisor mode qualification enabled</td></tr>
<tr><td>x</td><td>1</td><td>0</td><td>x</td><td>ASID qualification enabled; no mode qualification</td></tr>
<tr><td>x</td><td>1</td><td>1</td><td>0</td><td>ASID and user mode qualification enabled</td></tr>
<tr><td>x</td><td>1</td><td>1</td><td>1</td><td>ASID and supervisor mode qualification enabled</td></tr>
</table>
</td></tr>
</table>

34.3.8 Address Breakpoint Registers (ABLR/ABLR1, ABHR/ABHR1)

The ABLR, ABLR1, ABHR and ABHR1 define regions in the processor’s data address space that can act as part of the trigger. These register values are compared with the address for each transfer on the processor’s high-speed local bus. The trigger definition register (TDR) identifies the trigger as one of three cases:

- Identically the value in ABLR
- Inside the range bound by ABLR and ABHR inclusive
- Outside that same range

XTDR determines the same for ABLR1 and ABHR1.

ABLR, ABLR1, ABHR and ABHR1 are accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command.

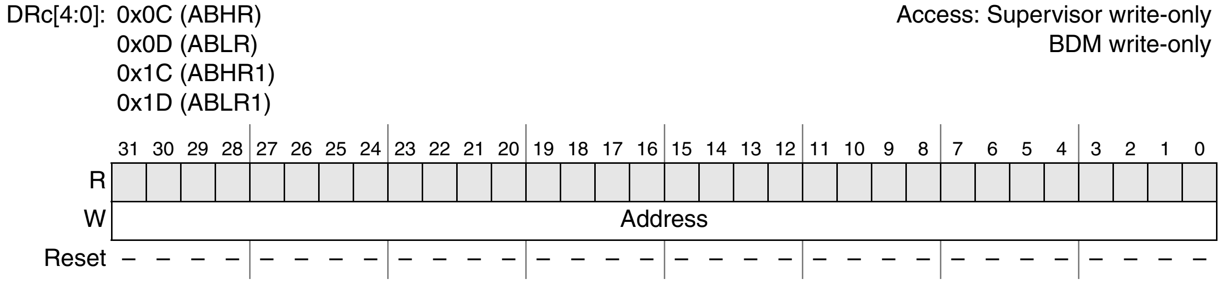


Figure 34-11. Address Breakpoint Registers (ABLR, ABHR, ABLR1, ABHR1)

Table 34-15. ABLR and ABLR1 Field Description

Field	Description
31–0 Address	Low Address. Holds the 32-bit address marking the lower bound of the address breakpoint range. Breakpoints for specific single addresses are programmed into ABLR or ABLR1.

Table 34-16. ABHR and ABHR1 Field Description

Field	Description
31–0 Address	High Address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

34.3.9 Data Breakpoint and Mask Registers (DBR/DBR1, DBMR/DBMR1)

The data breakpoint registers (DBR/DBR1), specify data patterns used as part of the trigger into debug mode. DBR*n* bits are masked by setting corresponding DBMR bits, as defined in TDR.

DBR, DBR1, DBMR, and DBMR1 are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command.

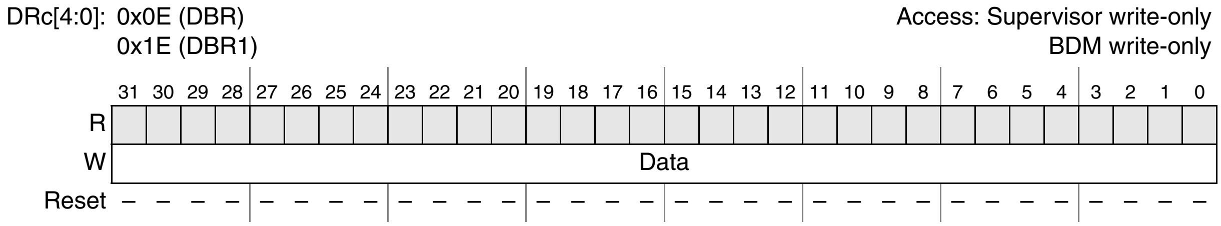
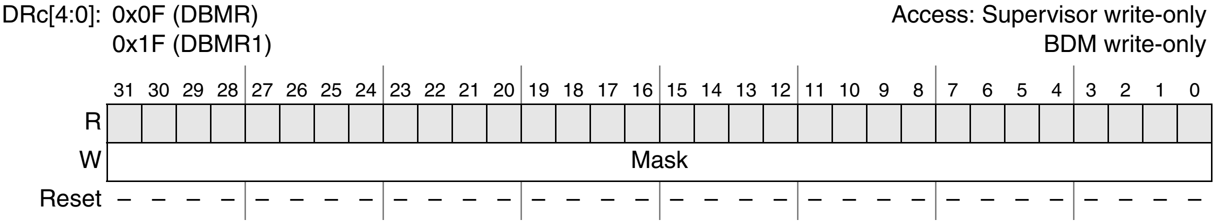


Figure 34-12. Data Breakpoint Registers (DBR, DBR1)

Table 34-17. DBR, DBR1 Field Descriptions

Field	Description
31–0 Data	Data Breakpoint Value. Contains the value to be compared with the data value from the processor’s local bus as a breakpoint trigger.



DRc[4:0]: 0x0F (DBMR)
0x1F (DBMR1)

Access: Supervisor write-only
BDM 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
R																																
W	Mask																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 34-13. Data Breakpoint Mask Registers (DBMR, DBMR1)

Table 34-18. DBMR, DBMR1 Field Descriptions

Field	Description
31–0 Mask	Data Breakpoint Mask. The 32-bit mask for the data breakpoint trigger. Clearing a DBMR bit allows the corresponding DBR bit to be compared to the appropriate bit of the processor's local data bus. Setting a DBMR bit causes that bit to be ignored.

The DBRs support aligned and misaligned references. Table 34-19 shows relationships between processor address, access size, and location within the 32-bit data bus.

Table 34-19. Address, Access Size, and Operand Data Location

Address[1:0]	Access Size	Operand Location
00	Byte	D[31:24]
01	Byte	D[23:16]
10	Byte	D[15:8]
11	Byte	D[7:0]
0x	Word	D[31:16]
1x	Word	D[15:0]
xx	Longword	D[31:0]

34.3.10 PC Breakpoint ASID Register (PBASID)

Each PC breakpoint register (PBR0–3) specifies an instruction address that can be used to trigger a breakpoint. To support debugging in a virtual environment, an ASID can optionally be associated with the instruction address in the PC breakpoint registers. The optional specification of an ASID value is made using PBASID and its exact inclusion within the breakpoint specification defined by the PBAC.

PBASID contains one 8-bit ASID values for each PC breakpoint register, as described in Table 34-20, which allows each PC breakpoint register to be associated with a unique virtual address and process.

DRc[4:0]: 0x14 (PBASID)

Access: Supervisor read/write
BDM 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
R W	PBA3SID								PBA2SID								PBA1SID								PBA0SID							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-14. PC Breakpoint ASID Register (PBASID)

Table 34-20. PBASID Field Descriptions

Field	Description
31–24 PBR3ASID	Corresponds to the ASID associated with PBR3.
23–16 PBR2ASID	Corresponds to the ASID associated with PBR2.
15–8 PBR1ASID	Corresponds to the ASID associated with PBR1.
7–0 PBR0ASID	Corresponds to the ASID associated with PBR0.

34.3.11 Extended Trigger Definition Register (XTDR)

The XTDR configures the operation of the hardware breakpoint logic that corresponds with the ABHR1/ABLR1/AATR1, DBR1/DBMR1 registers within the debug module. In conjunction with the TDR and its associated debug registers, XTDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as a one- or two-level triggers. TDR[31–16] or XTDR[31–16] bits define second-level triggers, and bits 15–0 define first-level triggers.

NOTE

The debug module has no hardware interlocks; so to prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR and XTDR (by clearing TDR[29,13] and XTDR[29,13]) before defining triggers.

A write to XTDR clears the CSR trigger status bits, CSR[BSTAT]. XTDR is accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command.

Section 34.3.11.1, “Resulting Set of Possible Trigger Combinations,” describes how to handle multiple breakpoint conditions.

DRc[4:0]: 0x17 (XTDR)

Access: Supervisor write-only
BDM write-only

Second Level Trigger

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R																
W	0	0	L2EBL	L2ED							L2DI	L2EA			0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

First Level Trigger

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	0	0	L1EBL	L1ED							L1DI	L1EA			0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-15. Extended Trigger Definition Register (XTDR)

Table 34-21. XTDR Field Descriptions

Field	Description
31–30	Reserved, must be cleared.
29 L2EBL	Enable level 2 breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 2 breakpoints 1 Enables all level 2 breakpoint triggers
28–22 L2ED	Enable level 2 data breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all ED bits disables data breakpoints.

TDR Bit	Description
28	Data longword. Entire processor's local data bus.
27	Lower data word.
26	Upper data word.
25	Lower lower data byte. Low-order byte of the low-order word.
24	Lower middle data byte. High-order byte of the low-order word.
23	Upper middle data byte. Low-order byte of the high-order word.
22	Upper upper data byte. High-order byte of the high-order word.

Field	Description
21 L2DI	Level 2 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents. 0 No inversion 1 Invert data breakpoint comparators.

Table 34-21. XTDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20–18
L2EA</td><td>Enable level 2 address breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.</td></tr>
<tr><td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.</td></tr>
<tr><td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR1.</td></tr>
</table></td></tr>
<tr><td>17–14</td><td>Reserved, must be cleared.</td></tr>
<tr><td>13
L1EBL</td><td>Enable level 1 breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 1 breakpoints
1 Enables all level 1 breakpoint triggers</td></tr>
<tr><td>12–6
L1ED</td><td>Enable level 1 data breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all L1ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>12</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>11</td><td>Lower data word.</td></tr>
<tr><td>10</td><td>Upper data word.</td></tr>
<tr><td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>5
L1DI</td><td>Level 1 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
</table>

Table 34-21. XTDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>4–2
L1EA</td><td>Enable level 1 address breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.</td></tr>
<tr><td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.</td></tr>
<tr><td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR1.</td></tr>
</table></td></tr>
<tr><td>1–0</td><td>Reserved, must be cleared.</td></tr>
</table>

34.3.11.1 Resulting Set of Possible Trigger Combinations

The resulting set of possible breakpoint trigger combinations consist of the following options where || denotes logical OR, && denotes logical AND, and {} denotes an optional additional trigger term:

One-level triggers of the form:

```
if       (PC_breakpoint)
if       (PC_breakpoint    || Address_breakpoint{&&  Data_breakpoint})
if       (PC_breakpoint    || Address_breakpoint{&&  Data_breakpoint}
                           || Address1_breakpoint {&&  Data1_breakpoint})

if (Address_breakpoint     {&&  Data_breakpoint})
if ((Address_breakpoint    {&&  Data_breakpoint})
                           || (Address1_breakpoint{&&  Data1_breakpoint}))

if       (Address1_breakpoint        {&&  Data1_breakpoint})
```

Two-level triggers of the form:

```
if       (PC_breakpoint)
         then if            (Address_breakpoint{&&  Data_breakpoint})

if       (PC_breakpoint)
         then if            (Address_breakpoint{&&  Data_breakpoint}
                  ||        Address1_breakpoint{&&  Data1_breakpoint})

if       (PC_breakpoint)
         then if           (Address1_breakpoint{&&  Data1_breakpoint})

if       (Address_breakpoint        {&&  Data_breakpoint})
         then if           (Address1_breakpoint{&&  Data1_breakpoint})

if       (Address1_breakpoint       {&&  Data1_breakpoint})
         then if           (Address_breakpoint{&&  Data_breakpoint})

if       (Address_breakpoint        {&&  Data_breakpoint})
```

```
         then if          (PC_breakpoint)

if       (Address1_breakpoint       {&&  Data1_breakpoint})
         then if          (PC_breakpoint)

if       (Address_breakpoint        {&&  Data_breakpoint})
         then if          (PC_breakpoint
                  ||      Address1_breakpoint{&&  Data1_breakpoint})

if       (Address1_breakpoint       {&&  Data1_breakpoint})
         then if          (PC_breakpoint|| Address_breakpoint{&&  Data_breakpoint})
```

In this example, PC_breakpoint is the logical summation of the PBR0/PBMR, PBR1, PBR2, and PBR3 breakpoint registers; Address_breakpoint is a function of ABHR, ABLR, and AATR; Data_breakpoint is a function of DBR and DBMR; Address1_breakpoint is a function of ABHR1, ABLR1, and AATR1; and Data1_breakpoint is a function of DBR1 and DBMR1. In all cases, the data breakpoints can be included with an address breakpoint to further qualify a trigger event as an option.

34.4 Functional Description

34.4.1 Background Debug Mode (BDM)

The ColdFire family implements a low-level system debugger in the microprocessor in a dedicated hardware module. Communication with the development system is managed through a dedicated, high-speed serial command interface. Although some BDM operations, such as CPU register accesses, require the CPU to be halted, other BDM commands, such as memory accesses, can be executed while the processor is running.

BDM is useful because:

- In-circuit emulation is not needed, so physical and electrical characteristics of the system are not affected.
- BDM is always available for debugging the system and provides a communication link for upgrading firmware in existing systems.
- Provides high-speed cache downloading (500 Kbytes/sec), especially useful for flash programming
- Provides absolute control of the processor, and thus the system. This allows quick hardware debugging with the same tool set used for firmware development.

34.4.1.1 CPU Halt

Although most BDM operations can occur in parallel with CPU operations, unrestricted BDM operation requires the CPU to be halted. The sources that can cause the CPU to halt are listed below in order of priority:

1. A catastrophic fault-on-fault condition automatically halts the processor.

2. A hardware breakpoint trigger can generate a pending halt condition similar to the assertion of $\overline{\text{BKPT}}$. This type of halt is always first marked as pending in the processor, which samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point. See Section 34.4.2.1, “Theory of Operation”.
3. The execution of a HALT instruction immediately suspends execution. Attempting to execute HALT in user mode while CSR[UHE] is cleared generates a privilege violation exception. If CSR[UHE] is set, HALT can be executed in user mode. After HALT executes, the processor can be restarted by serial shifting a GO command into the debug module. Execution continues at the instruction after HALT.
4. The assertion of the $\overline{\text{BKPT}}$ input is treated as a pseudo-interrupt; asserting $\overline{\text{BKPT}}$ creates a pending halt postponed until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction; if a pending halt is detected, the processor suspends execution and enters the halted state.

The are two special cases involving the assertion of $\overline{\text{BKPT}}$:

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the $\overline{\text{BKPT}}$ input is asserted within eight cycles after $\overline{\text{RESET}}$ is negated, the processor enters the halt state, signaling halt status (0xF) on the PSTDDATA outputs. While the processor is in this state, all resources accessible through the debug module can be referenced. This is the only chance to force the processor into emulation mode through CSR[EMU].
- After system initialization, the processor’s response to the GO command depends on the set of BDM commands performed while it is halted for a breakpoint. Specifically, if the PC register was loaded, the GO command causes the processor to exit halted state and pass control to the instruction address in the PC, bypassing normal reset exception processing. If the PC was not loaded, the GO command causes the processor to exit halted state and continue reset exception processing.
- The ColdFire architecture also manages a special case of $\overline{\text{BKPT}}$ asserted while the processor is stopped by execution of the STOP instruction. For this case, the processor exits the stopped mode and enters the halted state, at which point all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction, which follows the STOP opcode.

The CSR[27–24] bits indicate the halt source, showing the highest priority source for multiple halt conditions. Debug module revisions A and B clear CSR[27–24] upon a read of the CSR, but revision C and D (in V4) do not. The debug GO command clears CSR[26–24].

HALT can be recognized by counting 0xFF occurrences on PSTDDATA. The count is necessary to determine between a possible data output value of 0xFF and the HALT condition. Because data always follows a marker (0x8, 0x9, 0xA, or 0xB), PSTDDATA can display no more than four data 0xFFs. Two such scenarios exist:

- A B marker occurs on the left nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]

  0xBF
  0xFF

0xFF
0xFF
0xF*X* (*X* indicates that the next PST value is guaranteed to not be 0xF)

- A B marker occurs on the right nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]

  0x*Y*B
  0xFF
  0xFF
  0xFF
  0xFF
  0x*XY* (*X* indicates that the PST value is guaranteed to not be 0xF, and *Y* indicates a PSTDDATA value that doesn't affect the 0xFF count).

Thus, a count of nine or more sequential 0xF PSTDDATA nibbles or five or more sequential 0xFF PSTDDATA bytes signifies the HALT condition.

34.4.1.2 BDM Serial Interface

When the CPU is halted and PSTDDATA reflects the halt status, the development system can send unrestricted commands to the debug module. The debug module implements a synchronous serial protocol using two inputs (DSCLK and DSI) and one output (DSO), where DSO is specified as a delay relative to the rising edge of the processor clock. See Table 34-2. The development system serves as the serial communication channel master and must generate DSCLK.

The serial channel operates at a frequency from DC to 1/5 of the PSTCLK frequency. The channel uses full-duplex mode, where data is sent and received simultaneously by master and slave devices. The transmission consists of 17-bit packets composed of a status/control bit and a 16-bit data word. As shown in Figure 34-16, all state transitions are enabled on a rising edge of the PSTCLK clock when DSCLK is high; DSI is sampled and DSO is driven.

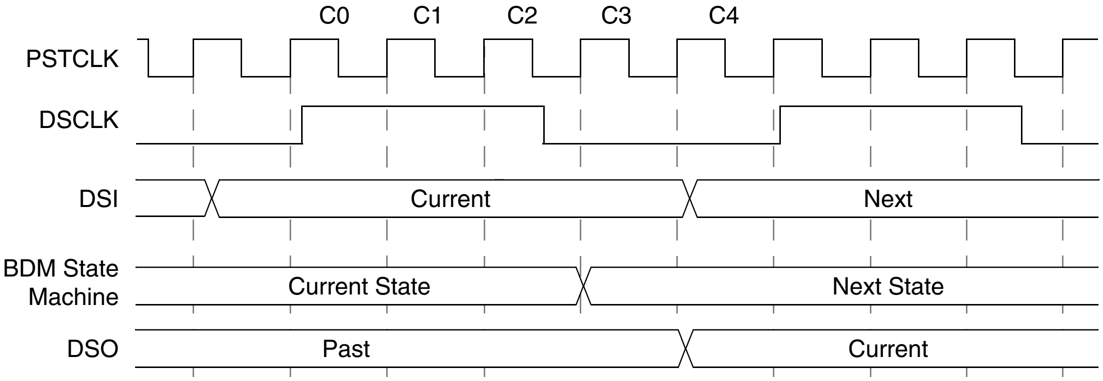


Figure 34-16. Maximum BDM Serial Interface Timing

DSCLK and DSI are synchronized inputs. DSCLK acts as a pseudo clock enable and is sampled, along with DSI, on the rising edge of PSTCLK. DSO is delayed from the DSCLK-enabled PSTCLK rising edge (registered after a BDM state machine state change). All events in the debug module's serial state machine are based on the PSTCLK rising edge. DSCLK must also be sampled low (on a positive edge of PSTCLK) between each bit exchange. The msb is sent first. Because DSO changes state based on an internally recognized rising edge of DSCLK, DSO cannot be used to indicate the start of a serial transfer. The development system must count clock cycles in a given transfer. C0–C4 are described as:

- C0: Set the state of the DSI bit
- C1: First synchronization cycle for DSI (DSCLK is high)
- C2: Second synchronization cycle for DSI (DSCLK is high)
- C3: BDM state machine changes state depending upon DSI and whether the entire input data transfer has been transmitted
- C4: DSO changes to next value

NOTE

> A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

34.4.1.3 Receive Packet Format

The basic receive packet consists of 16 data bits and 1 status bit

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	Data															

Figure 34-17. Receive BDM Packet

Table 34-22. Receive BDM Packet Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16
S</td><td>Status. Indicates the status of CPU-generated messages listed below. The not-ready response can be ignored unless a memory-referencing cycle is in progress. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.
<table>
<tr><th>S</th><th>Data</th><th>Message</th></tr>
<tr><td>0</td><td>xxxx</td><td>Valid data transfer</td></tr>
<tr><td>0</td><td>FFFF</td><td>Status OK</td></tr>
<tr><td>1</td><td>0000</td><td>Not ready with response; come again</td></tr>
<tr><td>1</td><td>0001</td><td>Error-Terminated bus cycle; data invalid</td></tr>
<tr><td>1</td><td>FFFF</td><td>Illegal Command</td></tr>
</table></td></tr>
<tr><td>15–0
Data</td><td>Data. Contains the message to be sent from the debug module to the development system. The response message is always a single word, with the data field encoded as shown above.</td></tr>
</table>

34.4.1.3.1 Transmit Packet Format

The basic transmit packet consists of 16 data bits and 1 reserved bit.

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
—	Data															

Figure 34-18. Transmit BDM Packet

Table 34-23. Transmit BDM Packet Field Description

Field	Description
16	Reserved, must be cleared.
15–0 Data	Data bits 15–0. Contains the data to be sent from the development system to the debug module.

34.4.1.3.2 BDM Command Format

All ColdFire family BDM commands include a 16-bit operation word followed by an optional set of one or more extension words.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Operation						0	R/W	Op Size		0	0	A/D	Register		
Extension Word(s)															

Figure 34-19. BDM Command Format

Table 34-24. BDM Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–10
Operation</td><td>Specifies the command. These values are listed in Table 34-25.</td></tr>
<tr><td>9</td><td>Reserved, must be cleared.</td></tr>
<tr><td>8
R/W</td><td>Direction of operand transfer.
0 Data is written to the CPU or to memory from the development system.
1 The transfer is from the CPU to the development system.</td></tr>
<tr><td>7–6
Op Size</td><td>Operand Data Size for Sized Operations. Addresses are expressed as 32-bit absolute values. A command performing a byte-sized memory read leaves the upper 8 bits of the response data undefined. Referenced data is returned in the lower 8 bits of the response.
<table>
<tr><th></th><th>Operand Size</th><th>Bit Values</th></tr>
<tr><td>00</td><td>Byte</td><td>8 bits</td></tr>
<tr><td>01</td><td>Word</td><td>16 bits</td></tr>
<tr><td>10</td><td>Longword</td><td>32 bits</td></tr>
<tr><td>11</td><td>Reserved</td><td>—</td></tr>
</table></td></tr>
<tr><td>5–4</td><td>Reserved, must be cleared.</td></tr>
</table>

Table 34-24. BDM Field Descriptions (continued)

Field	Description
3 A/D	Address/Data. Determines whether the register field specifies a data or address register. 0 Data register. 1 Address register.
2–0 Register	Contains the register number in commands that operate on processor registers. See Table 34-26.

34.4.1.3.3 Extension Words as Required

Some commands require extension words for addresses and/or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Longword accesses are forcibly longword-aligned and word accesses are forcibly word-aligned. Immediate data can be 1 or 2 words long. Byte and word data each requires a single extension word, while longword data requires two extension words.

Operands and addresses are transferred most-significant word first. In the following descriptions of the BDM command set, the optional set of extension words is defined as address, data, or operand data.

34.4.1.4 Command Sequence Diagrams

The command sequence diagram in Figure 34-20 shows serial bus traffic for commands. Each bubble represents a 17-bit bus transfer. The top half of each bubble indicates the data the development system sends to the debug module; the bottom half indicates the debug module's response to the previous development system commands. Command and result transactions overlap to minimize latency.

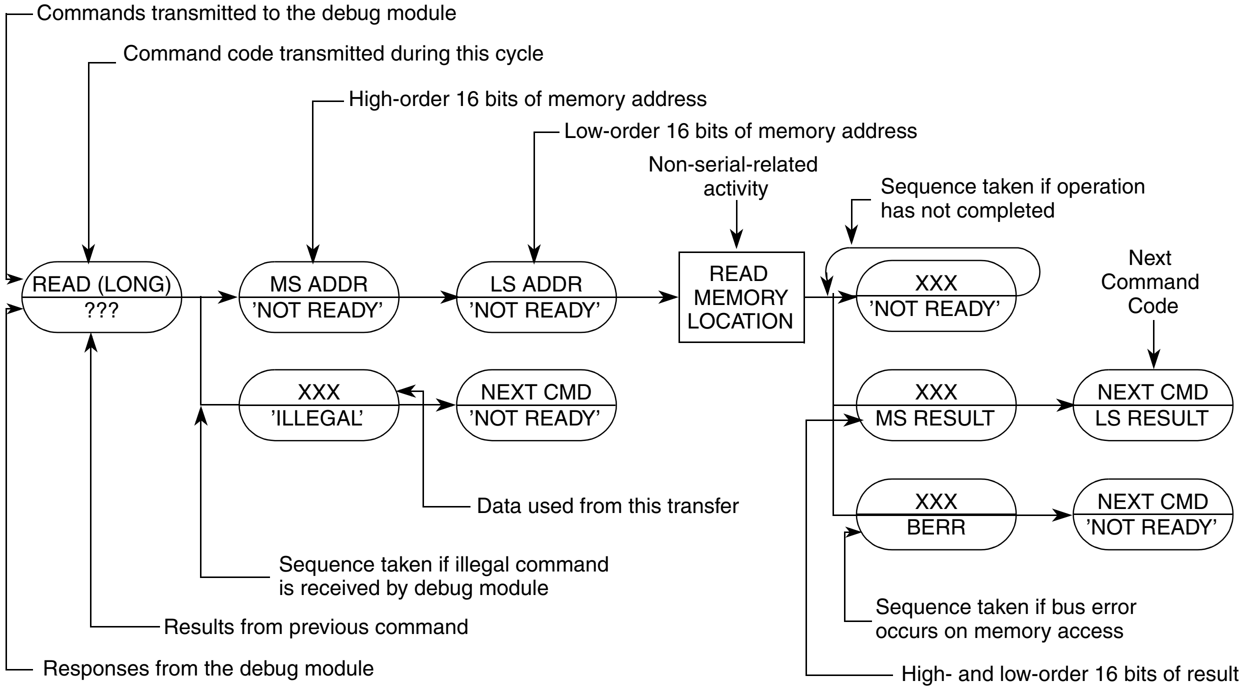

Commands transmitted to the debug module
Command code transmitted during this cycle
High-order 16 bits of memory address
Low-order 16 bits of memory address
Non-serial-related activity
Sequence taken if operation has not completed
READ (LONG)
???
MS ADDR
'NOT READY'
LS ADDR
'NOT READY'
READ MEMORY LOCATION
XXX
'NOT READY'
Next Command Code
XXX
'ILLEGAL'
NEXT CMD
'NOT READY'
XXX
MS RESULT
NEXT CMD
LS RESULT
XXX
BERR
NEXT CMD
'NOT READY'
Data used from this transfer
Sequence taken if illegal command is received by debug module
Results from previous command
Responses from the debug module
Sequence taken if bus error occurs on memory access
High- and low-order 16 bits of result

Figure 34-20. Command Sequence Diagram

The sequence is as follows:

- In cycle 1, the development system command is issued (READ in this example). The debug module responds with the low-order results of the previous command or a command complete status of the previous command, if no results are required.
- In cycle 2, the development system supplies the high-order 16 address bits. The debug module returns a not-ready response unless the received command is decoded as unimplemented, which is indicated by the illegal command encoding. If this occurs, the development system should retransmit the command.

NOTE

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

- In cycle 3, the development system supplies the low-order 16 address bits. The debug module always returns a not-ready response.
- At the completion of cycle 3, the debug module initiates a memory read operation. Any serial transfers that begin during a memory access return a not-ready response.
- Results are returned in the two serial transfer cycles after the memory access completes. For any command performing a byte-sized memory read operation, the upper 8 bits of the response data are undefined and the referenced data is returned in the lower 8 bits. The next command’s opcode is sent to the debug module during the final transfer. If a bus error terminates a memory or register access, error status (S = 1, DATA = 0x0001) returns instead of result data.

34.4.1.5 BDM Command Set

Table 34-25 summarizes the BDM command set. Subsequent sections contain detailed descriptions of each command. Issuing a BDM command when the processor is accessing debug module registers using the WDEBUG instruction causes undefined behavior. See Table 34-26 for register address encodings.

Table 34-25. BDM Command Summary

Command	Mnemonic	Description	CPU State[1]	Section/Page	Command (Hex)
Read A/D register	RAREG/ RDREG	Read the selected address or data register and return the results through the serial interface.	Halted	34.4.1.5.1/34-34	0x218 {A/D, Reg[2:0]}
Write A/D register	WAREG/ WDREG	Write the data operand to the specified address or data register.	Halted	34.4.1.5.2/34-34	0x208 {A/D, Reg[2:0]}
Read memory location	READ	Read the data at the memory location specified by the longword address.	Steal	34.4.1.5.3/34-35	0x1900—byte 0x1940—word 0x1980—lword
Write memory location	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	34.4.1.5.4/34-36	0x1800—byte 0x1840—word 0x1880—lword
Dump memory block	DUMP	Used with READ to dump large blocks of memory. An initial READ executes to set up the starting address of the block and to retrieve the first result. A DUMP command retrieves subsequent operands.	Steal	34.4.1.5.5/34-38	0x1D00—byte 0x1D40—word 0x1D80—lword
Fill memory block	FILL	Used with WRITE to fill large blocks of memory. An initial WRITE executes to set up the starting address of the block and to supply the first operand. A FILL command writes subsequent operands.	Steal	34.4.1.5.6/34-40	0x1C00—byte 0x1C40—word 0x1C80—lword
Resume execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	Halted	34.4.1.5.7/34-41	0x0C00
No operation	NOP	Perform no operation; may be used as a null command.	Parallel	34.4.1.5.8/34-42	0x0000
Output the current PC	SYNC_PC	Capture the current PC and display it on the PSTDDATA outputs.	Parallel	34.4.1.5.9/34-42	0x0001
Read control register	RCREG	Read the system control register.	Halted	34.4.1.5.11/34-44	0x2980
Write control register	WCREG	Write the operand data to the system control register.	Halted	34.4.1.5.14/34-47	0x2880
Read debug module register	RDMREG	Read the debug module register.	Parallel	34.4.1.5.15/34-48	0x2D {0x4[2] DRc[4:0]}
Write debug module register	WDMREG	Write the operand data to the debug module register.	Parallel	34.4.1.5.16/34-48	0x2C {0x4[2] DRc[4:0]}

[1] General command effect and/or requirements on CPU operation:
- Halted: The CPU must be halted to perform this command.
- Steal: Command generates bus cycles that can be interleaved with bus accesses.
- Parallel: Command is executed in parallel with CPU activity.

[2] 0x4 is a three-bit field.

Freescale reserves unassigned command opcodes. All unused command formats within any revision level perform a NOP and return the illegal command response.

The following sections describe the commands summarized in Table 34-25.

NOTE

> The BDM status bit (S) is 0 for normally completed commands. S is set for illegal commands, not-ready responses, and transfers with bus-errors. Section 34.4.1.2, "BDM Serial Interface," describes the receive packet format.

34.4.1.5.1 Read A/D Register (RAREG/RDREG)

Read the selected address or data register and return the 32-bit result. A bus error response is returned if the CPU core is not halted.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x1				0x8				A/D	Register		
Result	D[31:16]															
	D[15:0]															

Figure 34-21. RAREG/RDREG Command Format

Command Sequence:

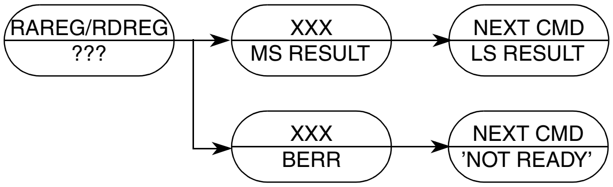


Figure 34-22. RAREG/RDREG Command Sequence

Operand Data: None

Result Data: The contents of the selected register are returned as a longword value, most-significant word first.

34.4.1.5.2 Write A/D Register (WAREG/WDREG)

The operand longword data is written to the specified address or data register. A write alters all 32 register bits. A bus error response is returned if the CPU core is not halted.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0x0				0x8				A/D	Register		
D[31:16]															
D[15:0]															

Figure 34-23. WAREG/WDREG Command Format

Command Sequence:

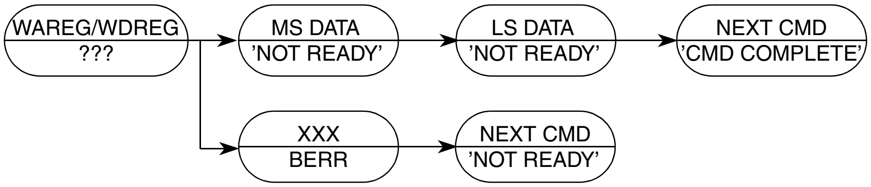


Figure 34-24. WAREG/WDREG Command Sequence

Operand Data: Longword data is written into the specified address or data register. The data is supplied most-significant word first.

Result Data: Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete.

34.4.1.5.3 Read Memory Location (READ)

Read data at the longword address. Address space is defined by BAAR[TT,TM]. Hardware forces low-order address bits to 0s for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command/Result Formats:

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	Command	0x1				0x9				0x0				0x0			
		A[31:16]															
		A[15:0]															
	Result	X	X	X	X	X	X	X	X	D[7:0]							
Word	Command	0x1				0x9				0x4				0x0			
		A[31:16]															
		A[15:0]															
	Result	D[15:0]															
Longword	Command	0x1				0x9				0x8				0x0			
		A[31:16]															
		A[15:0]															
	Result	D[31:16]															
		D[15:0]															

Figure 34-25. READ Command/Result Formats

Command Sequence:

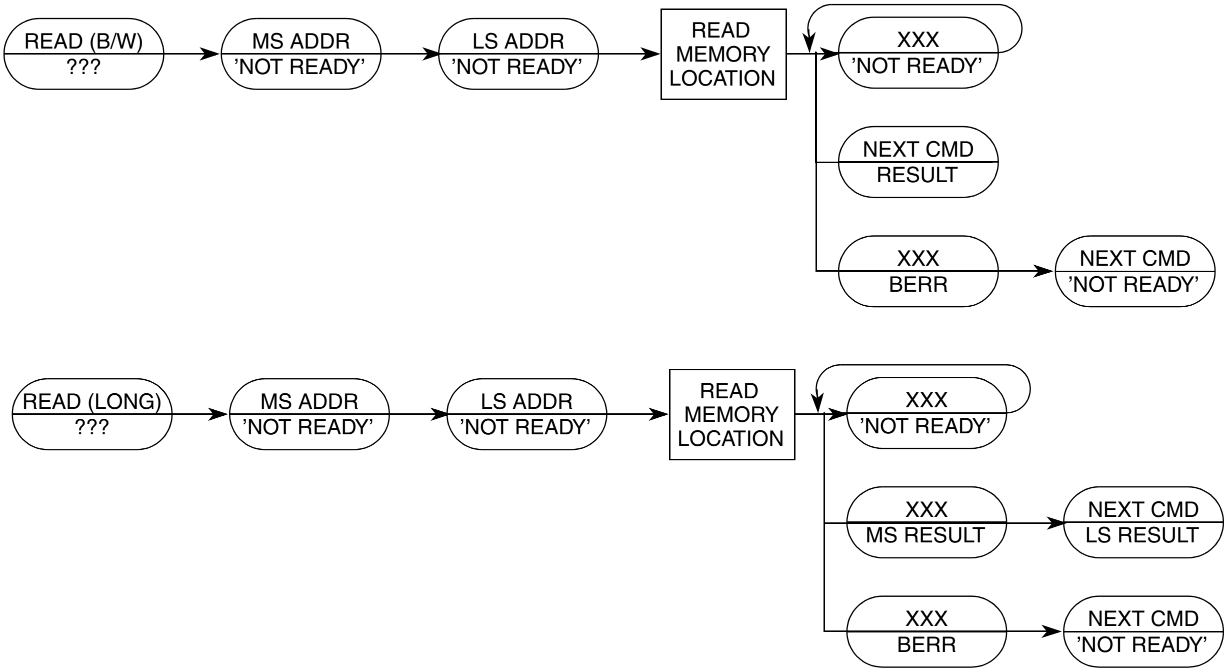


Figure 34-26. READ Command Sequence

Operand Data: The only operand is the longword address of the requested location.

Result Data: Word results return 16 bits of data; longword results return 32. Bytes are returned in the LSB of a word result; the upper byte is undefined. 0x0001 (S = 1) is returned if a bus error occurs.

34.4.1.5.4 Write Memory Location (WRITE)

Write data to the memory location specified by the longword address. BAAR[TT,TM] defines address space. Hardware forces low-order address bits to 0s for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command Formats:

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="4">Byte</td><td colspan="4">0x1</td><td colspan="4">0x8</td><td colspan="4">0x0</td><td colspan="4">0x0</td></tr>
<tr><td colspan="16">A[31:16]</td></tr>
<tr><td colspan="16">A[15:0]</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td colspan="8">D[7:0]</td></tr>
<tr><td rowspan="4">Word</td><td colspan="4">0x1</td><td colspan="4">0x8</td><td colspan="4">0x4</td><td colspan="4">0x0</td></tr>
<tr><td colspan="16">A[31:16]</td></tr>
<tr><td colspan="16">A[15:0]</td></tr>
<tr><td colspan="16">D[15:0]</td></tr>
<tr><td rowspan="5">Longword</td><td colspan="4">0x1</td><td colspan="4">0x8</td><td colspan="4">0x8</td><td colspan="4">0x0</td></tr>
<tr><td colspan="16">A[31:16]</td></tr>
<tr><td colspan="16">A[15:0]</td></tr>
<tr><td colspan="16">D[31:16]</td></tr>
<tr><td colspan="16">D[15:0]</td></tr>
</table>

Figure 34-27. WRITE Command Format

Command Sequence:

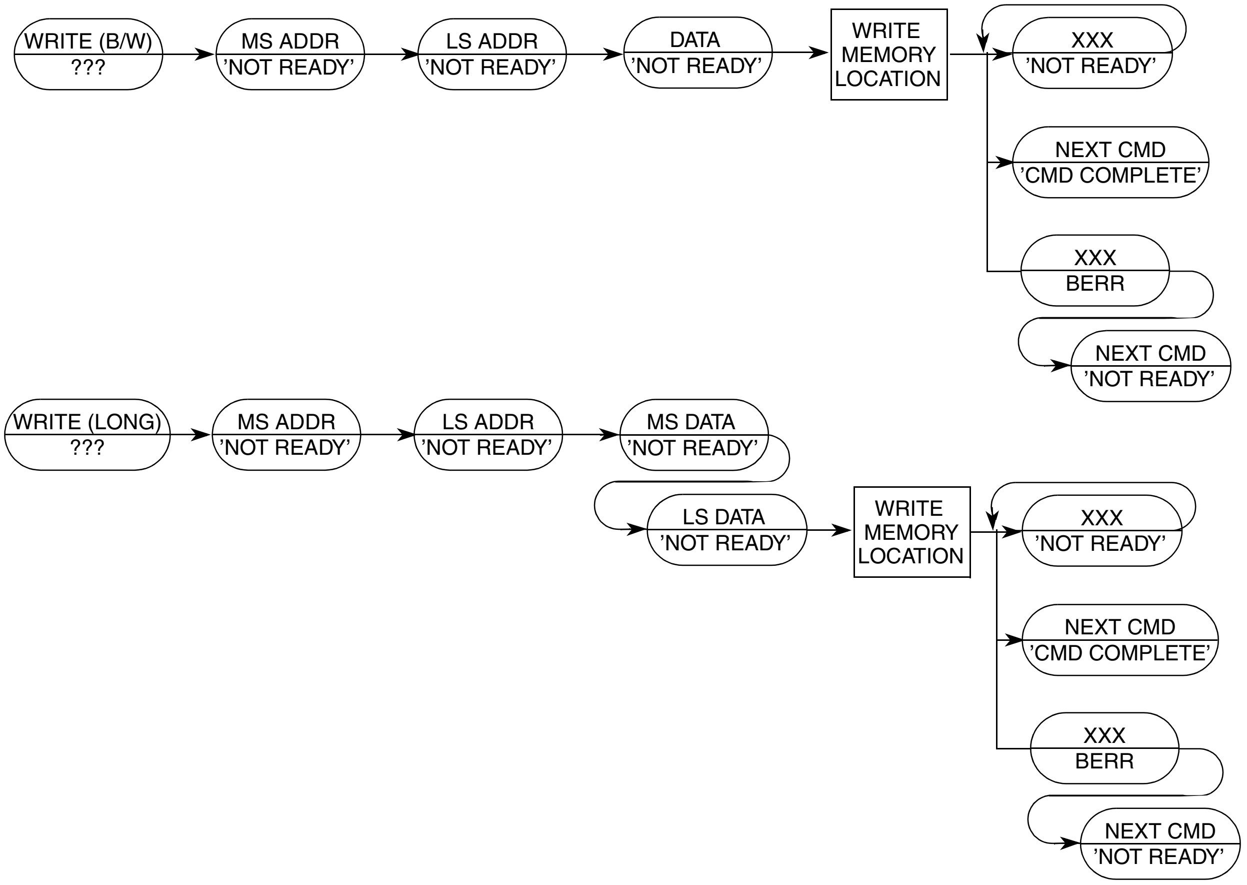


Figure 34-28. WRITE Command Sequence

Operand Data: This two-operand instruction requires a longword absolute address that specifies a location the data operand is written. Byte data is sent as a 16-bit word, justified in the LSB; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

Result Data: Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

34.4.1.5.5 Dump Memory Block (DUMP)

DUMP is used with the READ command to access large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. If an initial READ is not executed before the first DUMP, an illegal command response is returned. The DUMP command retrieves subsequent operands. The initial address increments by the operand size (1, 2, or 4) and saves in a temporary register. Subsequent DUMP commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in the temporary register.

NOTE

> DUMP does not check for a valid address; it is a valid command only when preceded by NOP, READ, or another DUMP command. Otherwise, an illegal command response is returned. NOP can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a DUMP command is processed, allowing the operand size to be dynamically altered.

Command/Result Formats:

		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	Command	0x1				0xD				0x0				0x0			
	Result	X	X	X	X	X	X	X	X	D[7:0]							
Word	Command	0x1				0xD				0x4				0x0			
	Result	D[15:0]															
Longword	Command	0x1				0xD				0x8				0x0			
	Result	D[31:16]															
		D[15:0]															

Figure 34-29. DUMP Command/Result Formats

Command Sequence:

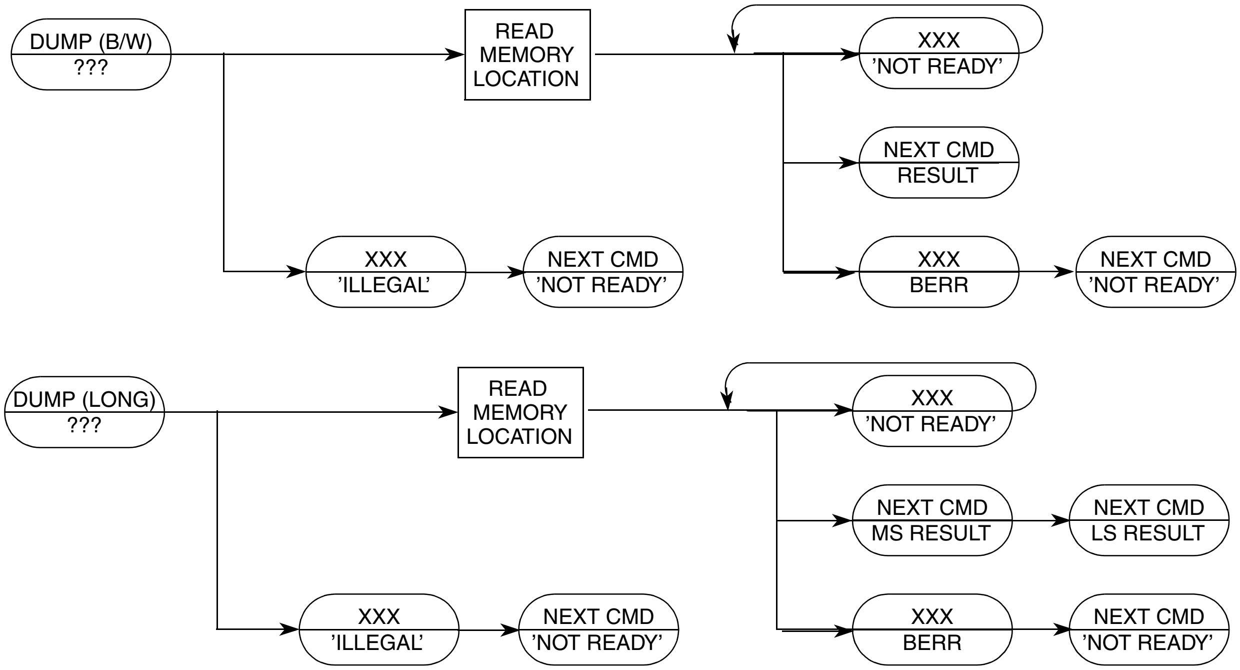


Figure 34-30. DUMP Command Sequence

Operand Data: None

Result Data: Requested data is returned as a word or longword. Byte data is returned in the least-significant byte of a word result. Word results return 16 bits of significant data; longword results return 32 bits. A value of 0x0001 (with S set) is returned if a bus error occurs.

34.4.1.5.6 Fill Memory Block (FILL)

A FILL command is used with the WRITE command to access large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. The FILL command writes subsequent operands. The initial address increments by the operand size (1, 2, or 4) and saves in a temporary register after the memory write. Subsequent FILL commands use this address, perform the write, increment it by the current operand size, and store the updated address in the temporary register.

If an initial WRITE is not executed preceding the first FILL command, the illegal command response is returned.

NOTE

The FILL command does not check for a valid address: FILL is a valid command only when preceded by another FILL, a NOP, or a WRITE command. Otherwise, an illegal command response is returned. The NOP command can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a FILL command is processed, allowing the operand size to be altered dynamically.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1				0xC				0x0				0x0			
	X	X	X	X	X	X	X	X	D[7:0]							
Word	0x1				0xC				0x4				0x0			
	D[15:0]															
Longword	0x1				0xC				0x8				0x0			
	D[31:16]															
	D[15:0]															

Figure 34-31. FILL Command Format

Command Sequence:

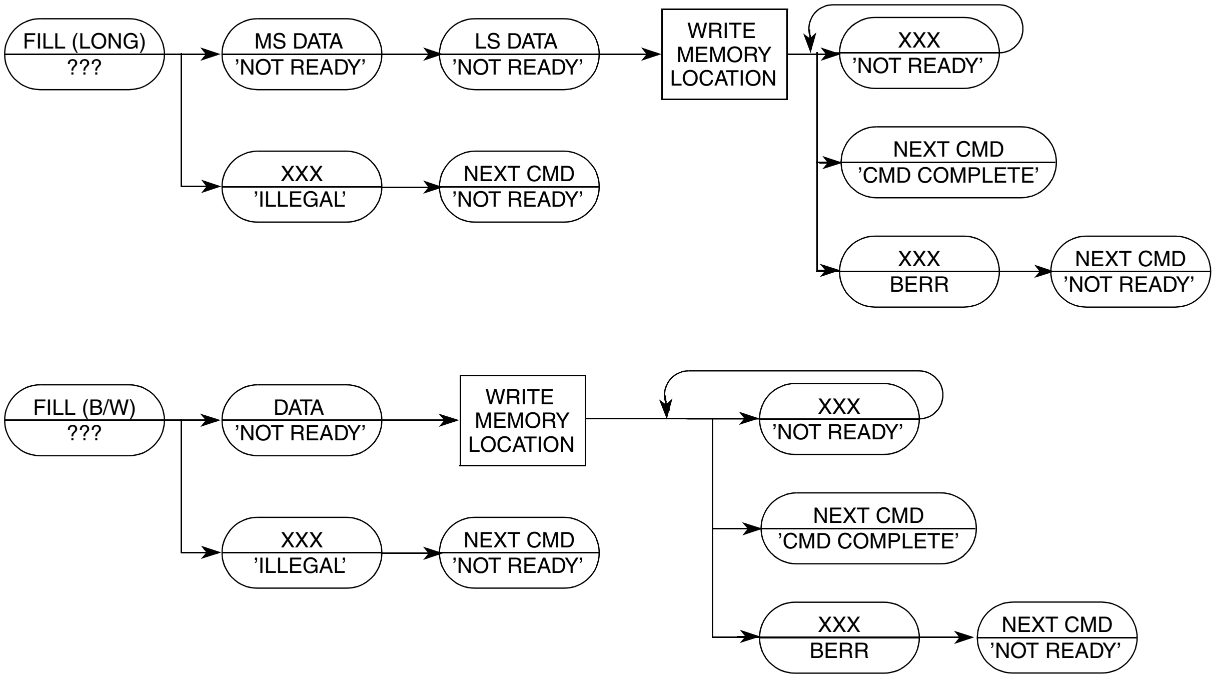


Figure 34-32. FILL Command Sequence

Operand Data: A single operand is data to be written to the memory location. Byte data is sent as a 16-bit word, justified in the least-significant byte; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

Result Data: Command complete status (0xFFFF) is returned when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

34.4.1.5.7 Resume Execution (GO)

The pipeline is flushed and refilled before normal instruction execution resumes. Prefetching begins at the current address in the PC and at the current privilege level. If any register (such as the PC or SR) is altered by a BDM command while the processor is halted, the updated value is used when prefetching resumes. If a GO command issues and the CPU is not halted, the command is ignored.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0				0xC				0x0				0x0			

Figure 34-33. GO Command Format

Command Sequence:

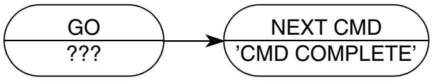


Figure 34-34. GO Command Sequence

Operand Data: None

Result Data: The command-complete response (0xFFFF) is returned during the next shift operation.

34.4.1.5.8 No Operation (NOP)

NOP performs no operation and may be used as a null command where required.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0				0x0				0x0				0x0			

Figure 34-35. NOP Command Format

Command Sequence:

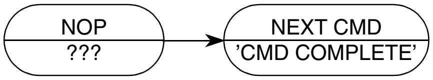


Figure 34-36. NOP Command Sequence

Operand Data: None

Result Data: The command-complete response, 0xFFFF (with S cleared), is returned during the next shift operation.

34.4.1.5.9 Synchronize PC to the PSTDDATA Lines (SYNC_PC)

The SYNC_PC command captures the current PC and displays it on the PSTDDATA outputs. After the debug module receives the command, it sends a signal to the ColdFire processor that the current PC must be displayed. The processor then forces an instruction fetch at the next PC with the address being captured in the DDATA logic under control of the CSR[BTB] bits. The specific sequence of PSTDDATA values is defined below:

1. Debug signals a SYNC_PC command is pending.
2. CPU completes the current instruction.
3. CPU forces an instruction fetch to the next PC, generates a PST equaling 0x5 value indicating a taken branch and signals the capture of DDATA.
4. The instruction address corresponding to the PC is captured.
5. The PST marker (0x9–0xB) is generated and displayed as defined by the CSR[BTB] bit followed by the captured PC address.

If the option to display ASID is enabled (CSR[OTE] = 1), the 8-bit ASID follows the address. That is, the PSTDDATA sequence is {0x5, Marker, Instruction Address, 0x8, ASID}, where the 0x8 is the marker for the ASID.

The SYNC_PC command can be used to dynamically access the PC for performance monitoring. The execution of this command is considerably less obtrusive to the real-time operation of an application than a HALT-CPU/READ-PC/RESUME command sequence.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0				0x0				0x0				0x1			

Figure 34-37. SYNC_PC Command Format

Command Sequence:

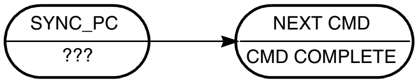


Figure 34-38. SYNC_PC Command Sequence

Operand Data:	None
Result Data:	Command complete status (0xFFFF) is returned when the register write is complete.

34.4.1.5.10 Force Transfer Acknowledge (FORCE_TA)

Debug revision D logic implements the new FORCE_TA serial BDM command to resolve a hung bus condition. In some system designs, references to certain unmapped memory addresses may cause the external bus to hang with no transfer acknowledge generated by any bus responders. The FORCE_TA forces generation of a transfer acknowledge signal.

There are two scenarios of interest: one caused by a processor access and the other caused by a BDM access. The following sequences identify the operations needed to break the hung bus condition:

- Bus hang caused by processor or external or internal alternate master:
  - Assert the breakpoint input to force a processor core halt.
  - If the bus hang was caused by a processor access, send in FORCE_TA commands until the processor is halted, as signaled by PST = 0xF. Due to pipeline and store buffer depths, many memory accesses may be queued up behind the access causing the bus hang. Repeated FORCE_TA commands eventually allow processing of all these pending accesses. As soon as the processor is halted, the system reaches a quiescent, controllable state.
  - If the hang was caused by another master, such as a DMA channel, the processor can halt immediately. In this case as well, multiple assertions of the FORCE_TA command may be required to terminate the alternate master's errant access.
- Bus hang caused by BDM access:
  - It is assumed the processor is already halted at the time of the errant BDM access. To resolve the hung bus, it is necessary to process four or more FORCE_TA commands, because the BDM command may have initiated a cache line access that fetches 4 longwords, each needing a unique transfer acknowledge.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0				0x0				0x0				0x2			

Figure 34-39. FORCE_TA Command

Command Sequence:

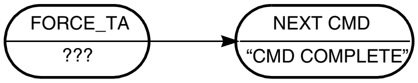


Figure 34-40. FORCE_TA Command Sequence

Operand Data: None

Result Data: The command complete response, 0xFFFF (with the status bit cleared), is returned during the next shift operation. This response indicates the FORCE_TA command was processed correctly and does not necessarily reflect the status of any internal bus.

34.4.1.5.11 Read Control Register (RCREG)

Read the selected control register and return the 32-bit result. Accesses to the processor/memory control registers are always 32 bits wide, regardless of register width. The second and third words of the command form a 32-bit address, which the debug module uses to generate a special bus cycle to access the specified control register. The 12-bit Rc field is the same the processor’s MOVEC instruction uses.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x9				0x8				0x0			
	0x0				0x0				0x0				0x0			
	0x0				Rc											
Result	D[31:16]															
	D[15:0]															

Figure 34-41. RCREG Command/Result Formats

Command Sequence:

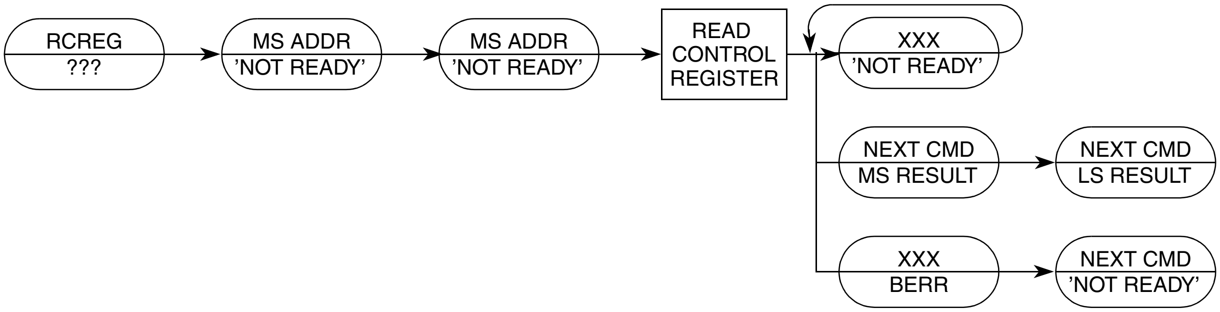


Figure 34-42. RCREG Command Sequence

Operand Data: The only operand is the 32-bit Rc control register select field.

Result Data: Control register contents are returned as a longword, most-significant word first. The implemented portion of registers smaller than 32 bits is guaranteed correct; other bits are undefined.

Rc encoding: See Table 34-26.

Table 34-26. Control Register Map

Rc	Register Definition
0x002	Cache Control Register (CACR)
0x003	Address Space Identifier (ASID)
0x004	Access Control Register (ACR0)
0x005	Access Control Register (ACR1)
0x006	Access Control Register (ACR2)
0x007	Access Control Register (ACR3)
0x008	MMU Base Address Register (MMUBAR)
0x009	RGPIO Base Address Register (RGPIOBAR)[1]
0x(0,1)80 – 0x(0,1)87	Data Registers 0–7 (0 = load, 1 = store)
0x(0,1)88 – 0x(0,1)8F	Address Registers 0–7 (0 = load, 1 = store) (A7 is user stack pointer)
0x800	Other Stack Pointer (OTHER_A7)
0x801	Vector Base Register (VBR)
0x804	MAC Status Register (MACSR)
0x805	MAC Mask Register (MASK)
0x806	MAC Accumulator 0 (ACC0)
0x807	MAC Accumulator 0,1 Extension Bytes (ACCEXT01)
0x808	MAC Accumulator 2,3 Extension Bytes (ACCEXT23)
0x809	MAC Accumulator 1 (ACC1)
0x80A	MAC Accumulator 2 (ACC2)

Table 34-26. Control Register Map (continued)

Rc	Register Definition
0x80B	MAC Accumulator 3 (ACC3)
0x80E	Status Register (SR)
0x80F	Program Register (PC)
0xC05	RAM Base Address Register (RAMBAR)

[1] If an RGPIO module is available on this device.

34.4.1.5.12 BDM Accesses of the Stack Pointer Registers (A7: SSP and USP)

The ColdFire core supports two unique stack pointer (A7) registers: the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two programmable-visible 32-bit registers does not uniquely identify one as the SSP and the other as the USP. Rather, the hardware uses one 32-bit register as the currently-active A7; the other is named the OTHER_A7. Therefore, the contents of the two hardware registers is a function of the operating mode of the processor:

```
if SR[S] = 1
         then     A7 = Supervisor Stack Pointer
                  OTHER_A7 = User Stack Pointer
         else     A7 = User Stack Pointer
                  OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports reads and writes to A7 and OTHER_A7 directly. It is the responsibility of the external development system to determine the mapping of A7 and OTHER_A7 to the two program-visible definitions (supervisor and user stack pointers), based on the SR[S] bit.

34.4.1.5.13 BDM Accesses of the EMAC Registers

The presence of rounding logic in the output datapath of the EMAC requires special care for BDM-initiated reads and writes of its programming model. In particular, any result rounding modes must be disabled during the read/write process so the exact bit-wise EMAC register contents are accessed.

For example, a BDM read of an accumulator (ACC*x*) must be preceded by two commands accessing the MAC status register, as shown in the following sequence:

```
BdmReadACCx (
         rcreg    macsr;            // read current macsr contents and  save
         wcreg    #0,macsr;         // disable all rounding modes
         rcreg    ACCx;             // read the desired accumulator
         wcreg    #saved_data,macsr;// restore the original macsr
)
```

Likewise, to write an accumulator register, the following BDM sequence is needed:

```
BdmWriteACCx (
         rcreg    macsr;            // read current macsr contents and  save
         wcreg    #0,macsr;         // disable all rounding modes
         wcreg    #data,ACCx;       // write the desired accumulator
         wcreg    #saved_data,macsr;// restore the original macsr
)
```

Additionally, writes to the accumulator extension registers must be performed after the corresponding accumulators are updated because a write to any accumulator alters the corresponding extension register contents.

For more information on saving and restoring the complete EMAC programming model, see Section 5.3.1.2, “Saving and Restoring the EMAC Programming Model.”

34.4.1.5.14 Write Control Register (WCREG)

The operand (longword) data is written to the specified control register. The write alters all 32 register bits. See the RCREG instruction description for the Rc encoding and for additional notes on writes to the A7 stack pointers and the EMAC programming model.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x8				0x8				0x0			
	0x0				0x0				0x0				0x0			
	0x0				Rc											
Result	D[31:16]															
	D[15:0]															

Figure 34-43. WCREG Command/Result Formats

Command Sequence:

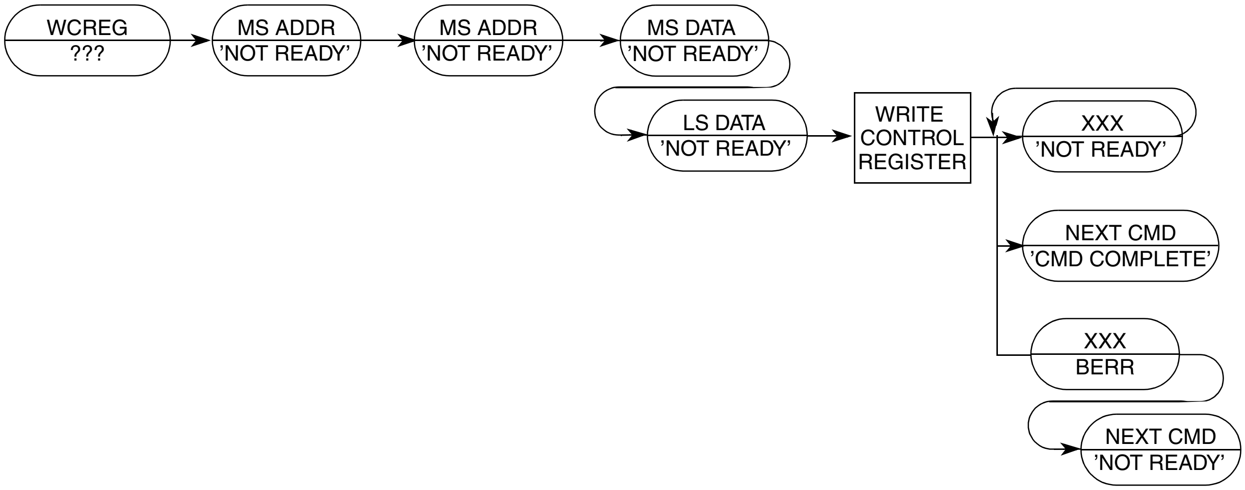


Figure 34-44. WCREG Command Sequence

Operand Data: This instruction requires two longword operands. The first selects the register to the operand data writes to; the second contains the data.

Result Data: Successful write operations return 0xFFFF. Bus errors on the write cycle are indicated by the setting of bit 16 in the status message and by a data pattern of 0x0001.

34.4.1.5.15 Read Debug Module Register (RDMREG)

Read the selected debug module register and return the 32-bit result. The only valid register selection for the RDMREG command is CSR (DRc=0x00).

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0xD				1	0	DRc					
Result	D[31:16]															
	D[15:0]															

Figure 34-45. RDMREG Command/Result Formats

Table 34-27 shows the definition of DRc encoding.

Table 34-27. Definition of DRc Encoding—Read

DRc[5:0]	Debug Register Definition	Mnemonic
0x00	Configuration/Status	CSR

Command Sequence:

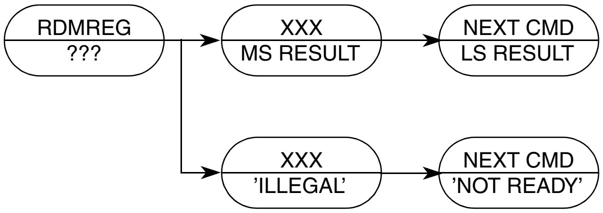


Figure 34-46. RDMREG Command Sequence

Operand Data: None

Result Data: The contents of the selected debug register are returned as a longword value. The data is returned most-significant word first.

34.4.1.5.16 Write Debug Module Register (WDMREG)

The operand (longword) data is written to the specified debug module register. All 32 bits of the register are altered by the write. DSCLK must be inactive while the debug module register writes from the CPU accesses are performed using the WDEBUG instruction.

Command Format:

Figure 34-47. WDMREG BDM Command Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0xC				1	0	DRc					
D[31:16]															
D[15:0]															

Table 34-5 shows the definition of the DRc write encoding.

Command Sequence:

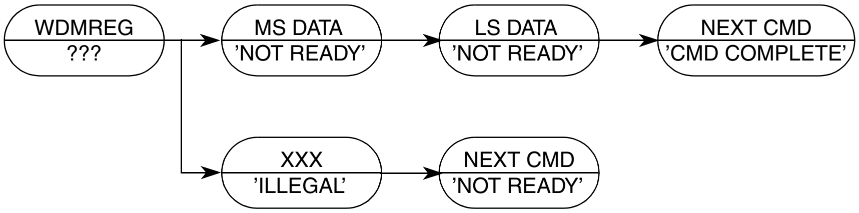


Figure 34-48. WDMREG Command Sequence

Operand Data: Longword data is written into the specified debug register. The data is supplied most-significant word first.

Result Data: Command complete status (0xFFFF) is returned when register write is complete.

34.4.2 Real-Time Debug Support

The ColdFire family provides support debugging real-time applications. For these types of embedded systems, the processor must continue to operate during debug. The foundation of this area of debug support is that while the processor cannot be halted to allow debugging, the system can generally tolerate the small intrusions of the BDM inserting instructions into the pipeline with minimal effect on real-time operation.

The debug module provides four types of breakpoints: PC with mask, PC without mask, operand address range, and data with mask. These breakpoints can be configured into one- or two-level triggers with the exact trigger response also programmable. The debug module programming model can be written from the external development system using the debug serial interface or from the processor's supervisor programming model using the WDEBUG instruction. Only CSR is readable using the external development system.

34.4.2.1 Theory of Operation

Breakpoint hardware can be configured through TDR[TCR] to respond to triggers by displaying PSTDDATA, initiating a processor halt, or generating a debug interrupt. As shown in Table 34-28, when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the PSTDDATA output port of the DDATA information when it is not displaying captured processor status, operands, or branch addresses. See Section 34.4.4.2, "Processor Stopped or Breakpoint State Change (PST = 0xE)."

Table 34-28. PSTDDATA Nibble/CSR[BSTAT] Breakpoint Response

PSTDDATA Nibble[1]	CSR[BSTAT][1]	Breakpoint Status
0000	0000	No breakpoints enabled
0010	0001	Waiting for level-1 breakpoint
0100	0010	Level-1 breakpoint triggered
1010	0101	Waiting for level-2 breakpoint
1100	0110	Level-2 breakpoint triggered

[1] Encodings not shown are reserved for future use.

The breakpoint status is also posted in the CSR. CSR[BSTAT] is cleared by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and a level-2 breakpoint is not enabled. Status is also cleared by writing to either TDR or XTDR to disable trigger options.

BDM instructions use the appropriate registers to load and configure breakpoints. As the system operates, a breakpoint trigger generates the response defined in TDR.

PC breakpoints are treated in a precise manner—exception recognition and processing are initiated before the excepting instruction executes. All other breakpoint events are recognized on the processor’s local bus, but are made pending to the processor and sampled like other interrupt conditions. As a result, these interrupts are imprecise.

In systems that tolerate the processor being halted, a BDM-entry can be used. With TDR[TRC] equals 01, a breakpoint trigger causes the core to halt (PST = 0xF).

If the processor core cannot be halted, the debug interrupt can be used. With this configuration, TDR[TRC] equals 10, breakpoint trigger becomes a debug interrupt to the processor, which is treated higher than the nonmaskable level-7 interrupt request. As with all interrupts, it is made pending until the processor reaches a sample point, which occurs once per instruction. Again, the hardware forces the PC breakpoint to occur before the targeted instruction executes and is precise. This is possible because the PC breakpoint is enabled when interrupt sampling occurs. For address and data breakpoints, reporting is considered imprecise, because several instructions may execute after the triggering address or data is detected.

As soon as the debug interrupt is recognized, the processor aborts execution and initiates exception processing. This event is signaled externally by the assertion of a unique PST value (PST = 0xD) for multiple cycles. The core enters emulator mode when exception processing begins. After the standard 8-byte exception stack is created, the processor fetches a unique exception vector from the vector table. Table 34-29 describes the two unique entries that distinguish PC breakpoints from other trigger events.

Table 34-29. Exception Vector Assignments

Vector Number	Vector Offset (Hex)	Stacked Program Counter	Assignment
12	0x030	Next	Non-PC-breakpoint debug interrupt
13	0x034	Next	PC-breakpoint debug interrupt

Refer to the *ColdFire Programmer's Reference Manual*. for more information.

In the case of a two-level trigger, the last breakpoint event determines the exception vector; however, if the second-level trigger is PC || Address {&& Data} (as shown in the last condition in the code example in Section 34.3.11.1, "Resulting Set of Possible Trigger Combinations"), the vector taken is determined by the first condition that occurs after the first-level trigger: vector 13 if PC occurs first or vector 12 if Address {&& Data} occurs first. If both occur simultaneously, the non-PC-breakpoint debug interrupt is taken (vector number 12).

Execution continues at the instruction address in the vector corresponding to the debug interrupt. The debug interrupt handler can use supervisor instructions to save the necessary context, such as the state of all program-visible registers into a reserved memory area.

During a debug interrupt service routine, all normal interrupt requests are evaluated and sampled once per instruction. If any exception occurs, the processor responds as follows:

1. It saves a copy of the current value of the emulator mode state bit and then exits emulator mode by clearing the actual state.
2. The fault status field (FS) in the next exception stack frame is set to 0010 to indicate the processor was in emulator mode when the interrupt occurred. See Section 3.3.3.1, "Exception Stack Frame Definition."
3. It passes control to the appropriate exception handler.
4. It executes an RTE instruction when the exception handler finishes. During the processing of the RTE, FS is reloaded from the system stack. If this bit field is set to 0010, the processor sets the emulator mode state and resumes execution of the original debug interrupt service routine. This is signaled externally by the generation of the PST value that originally identified the debug interrupt exception, that is, PST = 0xD.

Fault status encodings are listed in Table 3-7. The implementation of this debug interrupt handling fully supports the servicing of a number of normal interrupt requests during a debug interrupt service routine.

The emulator mode state bit is essentially changed to be a program-visible value, stored into memory during exception stack frame creation, and loaded from memory by the RTE instruction.

When debug interrupt operations complete, the RTE instruction executes and the processor exits emulator mode. After the debug interrupt handler completes execution, the external development system can use BDM commands to read the reserved memory locations.

In revisions B/B+ and C, the hardware inhibits generation of another debug interrupt during the first instruction after the RTE exits emulator mode. This behavior is consistent with the logic involving trace mode where the first instruction executes before another trace exception is generated. Thus, all hardware breakpoints are disabled until the first instruction after the RTE completes execution, regardless of the programmed trigger response.

34.4.2.2 Emulator Mode

Emulator mode facilitates non-intrusive emulator functionality. This mode can be entered in three different ways:

- Setting CSR[EMU] forces the processor into emulator mode. EMU is examined only if $\overline{\text{RSTI}}$ is negated and the processor begins reset exception processing. It can be set while the processor is halted before reset exception processing begins. See Section 34.4.1.1, "CPU Halt".
- A debug interrupt always puts the processor in emulation mode when debug interrupt exception processing begins.
- Setting CSR[TRC] forces the processor into emulation mode when trace exception processing begins.

While operating in emulation mode, the processor exhibits the following properties:

- Unmasked interrupt requests are serviced. The resulting interrupt exception stack frame has FS set appropriately (0010) to indicate the interrupt occurred while in emulator mode.
- If CSR[MAP] is set, all caching of memory and the SRAM module are disabled. All memory accesses are forced into a specially mapped address space signaled by TT equals 0x2, TM equals 0x5, or 0x6. This includes stack frame writes and vector fetch for the exception that forced entry into this mode.

The RTE instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry (0xD) and exit (0x7).

34.4.3 Concurrent BDM and Processor Operation

The debug module supports concurrent operation of the processor and most BDM commands. BDM commands may be executed while the processor is running, except these following operations that access processor/memory registers:

- Read/write address and data registers
- Read/write control registers

For BDM commands that access memory, the debug module requests the processor's local bus. The processor responds by stalling the instruction fetch pipeline and waiting for current bus activity to complete before freeing the local bus for the debug module to perform its access. After the debug module bus cycle, the processor reclaims the bus.

NOTE

Breakpoint registers must be carefully configured in a development system if the processor is executing. The debug module contains no hardware interlocks, so TDR and XTDR should be disabled while breakpoint registers are loaded, after which TDR and XTDR can be written to define the exact trigger. This prevents spurious breakpoint triggers.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug's registers (DSCLK must be inactive).

34.4.4 Real-Time Trace Support

Real-time trace, which defines the dynamic execution path and is also known as instruction trace, is a fundamental debug function. The ColdFire solution is to include a parallel output port providing encoded processor status and data to an external development system. This 8-bit port is partitioned into two consecutive 4-bit nibbles. Each nibble can either transmit information concerning the processor's execution status (PST) or debug data (DDATA). A PST marker and its data display are sent contiguously — the IDLE status (0x0) can appear anytime. As stated before, PST values and operands may appear on either nibble of PSTDDATA. The processor status may not be related to the current bus transfer, due to the decoupling FIFOs.

External development systems can use PSTDDATA outputs with an external image of the program to completely track the dynamic execution path. This tracking is complicated by any change in flow, where branch target address calculation is based on the contents of a program-visible register (variant addressing). PSTDDATA outputs can display the target address of such instructions in sequential nibble increments across multiple processor clock cycles, as described in Section 34.4.4.1, "Begin Execution of Taken Branch (PST = 0x5)". Four 32-bit storage elements form a FIFO buffer connecting the processor's high-speed local bus to the external development system through PSTDDATA[7:0]. The buffer captures branch target addresses and certain data values for eventual display on the PSTDDATA port, two nibbles at a time starting with the least significant bit (lsb).

Execution speed is affected only when three storage elements contain valid data to be dumped to the PSTDDATA port. This occurs only when two values are captured simultaneously in a read-modify-write operation. The core stalls until two FIFO entries are available.

Table 34-30 shows the encoding of these signals.

Table 34-30. Processor Status Encoding

PST[3:0]	Definition
0x0	Continue execution. Many instructions execute in one processor cycle. If an instruction requires more clock cycles, subsequent clock cycles are indicated by driving PSTDDATA outputs with this encoding.
0x1	Begin execution of one instruction. For most instructions, this encoding signals the first processor clock cycle of an instruction's execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions, generate different encodings.
0x2	Begin execution of two instructions. For superscalar instruction dispatches, this encoding signals the first clock cycle of the simultaneous instructions' execution.
0x3	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode. If the display of the ASID is enabled (CSR[OTE] = 1), the following occurs: The 8-bit ASID follows the instruction address; that is, the PSTDDATA sequence is {0x3, 0x5, marker, instruction address, 0x8, ASID}, where 0x8 is the ASID data marker. When the current ASID is loaded by the privileged MOVEC instruction, the ASID is displayed on PSTDDATA. The resulting PSTDDATA sequence for the MOVEC instruction is then {0x1, 0x8, ASID}, where the 0x8 is the data marker for the ASID.
0x4	Begin execution of PULSE and WDDATA instructions. PULSE defines logic analyzer triggers for debug and/or performance analysis. WDDATA lets the core write any operand (byte, word, or longword) directly to the PSTDDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled, followed by the appropriate marker, and then the data transfer on the PSTDDATA port. Transfer length depends on the WDDATA operand size.

Table 34-30. Processor Status Encoding (continued)

PST[3:0]	Definition
0x5	Begin execution of taken branch or SYNC_PC command issued. For some opcodes, a branch target address may be displayed on PSTDDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the PSTDDATA nibble that begins the data output. See Section 34.4.4.1, “Begin Execution of Taken Branch (PST = 0x5)”. Also indicates that the SYNC_PC command has been issued.
0x6	Begin execution of instruction plus a taken branch. The processor completes execution of a taken conditional branch instruction and simultaneously starts executing the target instruction. This is achieved through branch folding.
0x7	Begin execution of return from exception (RTE) instruction.
0x8– 0xB	Indicates the number of bytes to be displayed on the PSTDDATA port on subsequent clock cycles. The value is driven onto the PSTDDATA port one cycle before the data is displayed. 0x8 Begin 1-byte transfer on PSTDDATA. 0x9 Begin 2-byte transfer on PSTDDATA. 0xA Begin 3-byte transfer on PSTDDATA. 0xB Begin 4-byte transfer on PSTDDATA.
0xC	Normal exception processing. Exceptions that enter emulation mode (debug interrupt or optionally trace) generate a different encoding, as described below. Because the 0xC encoding defines a multiple-cycle mode, PSTDDATA outputs are driven with 0xC until exception processing completes.
0xD	Emulator mode exception processing. Displayed during emulation mode (debug interrupt or optionally trace). Because this encoding defines a multiple-cycle mode, PSTDDATA outputs are driven with 0xD until exception processing completes.
0xE	A breakpoint state change causes this encoding to assert for one cycle only followed by the trigger status value. If the processor stops waiting for an interrupt, the encoding is asserted for multiple cycles. See Section 34.4.4.2, “Processor Stopped or Breakpoint State Change (PST = 0xE).”
0xFF	Processor is halted. Because this encoding defines a multiple-cycle mode, the PSTDDATA outputs display 0xF until the processor is restarted or reset. See Section 34.4.1.1, “CPU Halt”.

34.4.4.1 Begin Execution of Taken Branch (PST = 0x5)

PST is 0x5 when a taken branch is executed. For some opcodes, a branch target address may be displayed on PSTDDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, which is indicated by the PST marker value immediately preceding the PSTDDATA nibble that begins the data output.

Multiple byte DDATA values are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches that use a variant addressing mode (RTE and RTS instructions, JMP and JSR instructions using address register indirect or indexed addressing modes, and all exception vectors).

The simplest example of a branch instruction using a variant address is the compiled code for a C language case statement. Typically, the evaluation of this statement uses the variable of an expression as an index into a table of offsets, where each offset points to a unique case within the structure. For such change-of-flow operations, the ColdFire processor uses the debug pins to output the following sequence of information on two successive processor clock cycles:

1. Use PSTDDATA (0x5) to identify that a taken branch is executed.

2. Signal the target address to be displayed sequentially on the PSTDDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.Using the PSTB, o
3. The new target address is optionally available on subsequent cycles using the PSTDDATA port. The number of bytes of displayed on this port is configurable (2, 3, or 4 bytes, where the PSTDDATA encoding is 0x9, 0xA, and 0xB, respectively).

Another example of a variant branch instruction would be a JMP (A0) instruction. Figure 34-49 shows the PSTDDATA outputs that indicate a JMP (A0) execution, assuming the CSR was programmed to display the lower 2 bytes of an address.

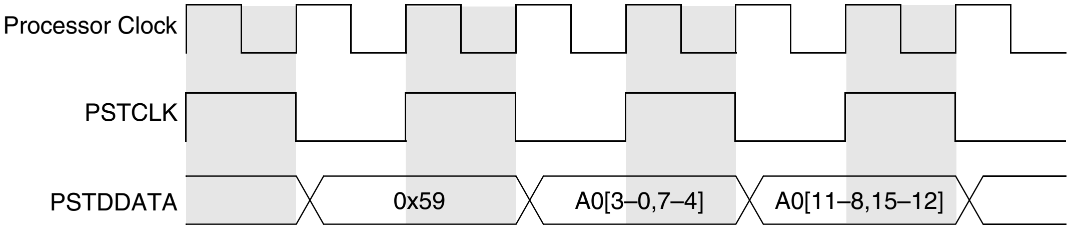


Figure 34-49. Example JMP Instruction Output on PSTDDATA

PSTDDATA is driven two nibbles at a time with a 0x59; 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Therefore, the subsequent 4 nibbles display the lower two bytes of address register A0 in least-to-most-significant nibble order. The PSTDDATA output after the JMP instruction continues with the next instruction.

34.4.4.2 Processor Stopped or Breakpoint State Change (PST = 0xE)

The 0xE encoding is generated either as a one- or multiple-cycle issue as follows:

- When the core is stopped by a STOP instruction, this encoding appears in multiple-cycle format. The ColdFire processor remains stopped until an interrupt occurs; thus, PSTDDATA outputs display 0xE until stopped mode is exited. PSTB only stores two consecutive packets of 0x1E.
- When a breakpoint status change is to be output on PSTDDATA, 0xE is displayed for one cycle, followed immediately with the 4-bit value of the current trigger status, where the trigger status is left justified rather than in the CSR[BSTAT] description. Section 34.3.2, “Configuration/Status Register (CSR),” shows that status is right justified. That is, the displayed trigger status on PSTDDATA after a single 0xE is as follows:
  - — 0x0 = no breakpoints enabled
  - — 0x2 = waiting for level-1 breakpoint
  - — 0x4 = level-1 breakpoint triggered
  - — 0xA = waiting for level-2 breakpoint
  - — 0xC = level-2 breakpoint triggered

Thus, 0xE can indicate multiple events, based on the next value, as Table 34-31 shows.

Table 34-31. 0xE Status Posting

PSTDDATA Stream Includes	Result
{0xE, 0x2}	Breakpoint state changed to waiting for level-1 trigger
{0xE, 0x4}	Breakpoint state changed to level-1 breakpoint triggered
{0xE, 0xA}	Breakpoint state changed to waiting for level-2 trigger
{0xE, 0xC}	Breakpoint state changed to level-2 breakpoint triggered
{0xE, 0xE}	Stopped mode.

34.4.4.3 Processor Halted (PST = 0xF)

PST is 0xF when the processor is halted (see Section 34.4.1.1, “CPU Halt”). Because this encoding defines a multiple-cycle mode, the PSTDDATA outputs display 0xF until the processor is restarted or reset. Therefore, PSTDDATA[7:0] continuously are 0xFF. PSTB only stores two consecutive packets of 0x1F.

NOTE

> HALT can be distinguished from a data output 0xFF by counting 0xFF occurrences on PSTDDATA. Because data always follows a marker (0x8, 0x9, 0xA, or 0xB), the longest occurrence in PSTDDATA of 0xFF in a data output is four.

Two scenarios exist for data 0xFFFF_FFFF:

- The B marker occurs on the most-significant nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]
  0xBF
  0xFF
  0xFF
  0xFF
  0xF*X* (*X* indicates that the next PST value is guaranteed to not be 0xF.)

- The B marker occurs on the least-significant nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]
  0xYB
  0xFF
  0xFF
  0xFF
  0xFF
  0x*XY* (*X* indicates the PST value is guaranteed not to be 0xF, and *Y* signifies a PSTDDATA value that doesn’t affect the 0xFF count.)

NOTE

As the result of the above, a count of nine or more sequential 0xF PSTDDATA nibbles, or five or more sequential 0xFF PSTDDATA bytes indicates the HALT condition.

34.4.5 Processor Status, Debug Data Definition

This section specifies the ColdFire processor and debug module's generation of the processor status/debug data (PSTDDATA) output on an instruction basis. In general, the PSTDDATA output for an instruction is defined as follows:

PSTDDATA = 0x1, {[0x89B], operand}

where the {...} definition is optional operand information defined by the setting of the CSR.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x8, 0x9, or 0xB} identifies the size and presence of valid data to follow on the PSTDDATA output {1, 2, or 4 bytes}. Additionally, for certain change-of-flow branch instructions, CSR[BTB] provides the capability to display the target instruction address on the PSTDDATA output {2, 3, or 4 bytes} using a PST value of {0x9, 0xA, or 0xB}. Addresses use the markers x0D, x0E, or 0xF to store 2, 3, or 4 bytes of address packets with address shifted right by 1 bit.

34.4.5.1 User Instruction Set

Table 34-32 shows the PSTDDATA specification for user-mode instructions. Rn represents any {Dn, An} register. In this definition, the y suffix generally denotes the source, and x denotes the destination operand. For a given instruction, the optional operand data is displayed only for those effective addresses referencing memory.

Table 34-32. PSTDDATA Specification for User-Mode Instructions

Instruction	Operand Syntax	PSTDDATA Nibble
add.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
add.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
adda.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source operand}
addi.l	#<data>,Dx	PSTDDATA = 0x1
addq.l	#<data>,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
addx.l	Dy,Dx	PSTDDATA = 0x1
and.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
and.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
andi.l	#<data>,Dx	PSTDDATA = 0x1
asl.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
asr.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
bcc.{b,w}		if taken, then PSTDDATA = 0x5, else PSTDDATA = 0x1

Table 34-32. PSTDDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PSTDDATA Nibble
bchg.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bchg.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bclr.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bclr.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bitrev.l	Dx	PSTDDATA = 0x1
bra.{b,w}		PSTDDATA = 0x5
bset.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bset.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bsr.{b,w}		PSTDDATA = 0x5, {0xB, destination operand}
btst.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source operand}
btst.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source operand}
byterev.l	Dx	PSTDDATA = 0x1
clr.b	<ea>x	PSTDDATA = 0x1, {0x8, destination operand}
clr.l	<ea>x	PSTDDATA = 0x1, {0xB, destination operand}
clr.w	<ea>x	PSTDDATA = 0x1, {0x9, destination operand}
cmp.b	<ea>y,Dx	PSTDDATA = 0x1, {0x8, source operand}
cmp.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
cmp.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
cmpa.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source operand}
cmpa.w	<ea>y,Ax	PSTDDATA = 0x1, {0x9, source operand}
cmpi.b	#<data>,Dx	PSTDDATA = 0x1
cmpi.l	#<data>,Dx	PSTDDATA = 0x1
cmpi.w	#<data>,Dx	PSTDDATA = 0x1
divs.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
divs.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
divu.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
divu.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
eor.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
eori.l	#<data>,Dx	PSTDDATA = 0x1
ext.l	Dx	PSTDDATA = 0x1
ext.w	Dx	PSTDDATA = 0x1
extb.l	Dx	PSTDDATA = 0x1
illegal		PSTDDATA = 0x1[1]

Table 34-32. PSTDDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PSTDDATA Nibble
jmp	<ea>y	PSTDDATA = 0x5, {[0x9AB], target address}[2]
jsr	<ea>y	PSTDDATA = 0x5, {[0x9AB], target address},{0xB, destination operand}[2]
lea.l	<ea>y,Ax	PSTDDATA = 0x1
link.w	Ay,#<displacement>	PSTDDATA = 0x1, {0xB, destination operand}
lsl.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
lsr.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
mov3q.l	#<data>,<ea>x	PSTDDATA = 0x1, {0xB,destination operand}
move.b	<ea>y,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
move.l	<ea>y,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
move.w	<ea>y,<ea>x	PSTDDATA = 0x1, {0x9, source}, {0x9, destination}
move.w	CCR,Dx	PSTDDATA = 0x1
move.w	{Dy,#<data>},CCR	PSTDDATA = 0x1
movea.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source}
movea.w	<ea>y,Ax	PSTDDATA = 0x1, {0x9, source}
movem.l	#list,<ea>x	PSTDDATA = 0x1, {0xB, destination},... [3]
movem.l	<ea>y,#list	PSTDDATA = 0x1, {0xB, source},... [3]
moveq.l	#<data>,Dx	PSTDDATA = 0x1
muls.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
muls.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
mulu.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
mulu.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
mvs.b	<ea>y,Dx	PSTDDATA = 0x1, {0x8, source operand}
mvs.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
mvz.b	<ea>y,Dx	PSTDDATA = 0x1, {0x8, source operand}
mvz.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
neg.l	Dx	PSTDDATA = 0x1
negx.l	Dx	PSTDDATA = 0x1
nop		PSTDDATA = 0x1
not.l	Dx	PSTDDATA = 0x1
or.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
or.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
ori.l	#<data>,Dx	PSTDDATA = 0x1
pea.l	<ea>y	PSTDDATA = 0x1, {0xB, destination operand}

Table 34-32. PSTDDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PSTDDATA Nibble
pulse		PSTDDATA = 0x4
rems.l	<ea>y,Dw:Dx	PSTDDATA = 0x1, {0xB, source operand}
remu.l	<ea>y,Dw:Dx	PSTDDATA = 0x1, {0xB, source operand}
sats.l	Dx	PSTDDATA = 0x1
scc.b	Dx	PSTDDATA = 0x1
sub.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
sub.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
suba.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source operand}
subi.l	#<data>,Dx	PSTDDATA = 0x1
subq.l	#<data>,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
subx.l	Dy,Dx	PSTDDATA = 0x1
swap.w	Dx	PSTDDATA = 0x1
tas.b	<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
tpf		PST = 0x1
tpf.l	#<data>	PST = 0x1
tpf.w	#<data>	PST = 0x1
trap	#<data>	PSTDDATA = 0x1[1]
tst.b	<ea>x	PSTDDATA = 0x1, {0x8, source operand}
tst.l	<ea>y	PSTDDATA = 0x1, {0xB, source operand}
tst.w	<ea>y	PSTDDATA = 0x1, {0x9, source operand}
unlk	Ax	PSTDDATA = 0x1, {0xB, destination operand}
wddata.b	<ea>y	PSTDDATA = 0x4, {0x8, source operand}
wddata.l	<ea>y	PSTDDATA = 0x4, {0xB, source operand}
wddata.w	<ea>y	PSTDDATA = 0x4, {0x9, source operand}

[1] During normal exception processing, the PSTDDATA output is driven to a 0xCC indicating the exception processing state. The exception stack write operands, as well as the vector read and target address of the exception handler may also be displayed.

```
Exception Processing:
     PSTDDATA = 0xCC,
     {0xB,destination},                   // stack frame
     {0xB,destination},                   // stack frame
     {0xB,source},                        // vector read
     PSTDDATA = 0x5,{[0x9AB],target}      // handler PC
```

The PSTDDATA specification for the reset exception is shown below:

```
Exception Processing:
     PSTDDATA = 0xCC,
     PSTDDATA = 0x5,{[0x9AB],target}     //  handler PC
```

The initial references at address 0 and 4 are never captured nor displayed because these accesses are treated as instruction fetches.

For all types of exception processing, the PSTDDATA = 0xCC value is driven at all times, unless the PSTDDATA output is needed for one of the optional marker values or for the taken branch indicator (0x5).

[2] For JMP and JSR instructions, the optional target instruction address is displayed only for those effective address fields defining variant addressing modes. This includes the following <ea>x values: (An), (d16,An), (d8,An,Xi), (d8,PC,Xi).

[3] For move multiple instructions (MOVEM), the processor automatically generates line-sized transfers if the operand address reaches a 0-modulo-16 boundary and there are four or more registers to be transferred. For these line-sized transfers, the operand data is never captured nor displayed, regardless of the CSR value.
The automatic line-sized burst transfers are provided to maximize performance during these sequential memory access operations.

Table 34-33 shows the PSTDDATA specification for multiply-accumulate instructions.

Table 34-33. PSTDDATA Values for User-Mode Multiply-Accumulate Instructions

Instruction	Operand Syntax	PSTDDATA Nibble
mac.l	Ry,Rx,ACCx	PSTDDATA = 0x1
mac.l	Ry,Rx,<ea>y,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}
mac.w	Ry,Rx,ACCx	PSTDDATA = 0x1
mac.w	Ry,Rx,ea,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}
move.l	{Ry,#<data>},ACC*x*	PSTDDATA = 0x1
move.l	{Ry,#<data>},MACSR	PSTDDATA = 0x1
move.l	{Ry,#<data>},MASK	PSTDDATA = 0x1
move.l	{Ry,#<data>},ACCext01	PSTDDATA = 0x1
move.l	{Ry,#<data>},ACCext23	PSTDDATA = 0x1
move.l	ACCext01,Rx	PSTDDATA = 0x1
move.l	ACCext23,Rx	PSTDDATA = 0x1
move.l	ACCy,ACCx	PSTDDATA = 0x1
move.l	ACCy,Rx	PSTDDATA = 0x1
move.l	MACSR,CCR	PSTDDATA = 0x1
move.l	MACSR,Rx	PSTDDATA = 0x1

Table 34-33. PSTDDATA Values for User-Mode Multiply-Accumulate Instructions (continued)

Instruction	Operand Syntax	PSTDDATA Nibble
move.l	MASK,Rx	PSTDDATA = 0x1
msac.l	Ry,Rx,ACCx	PSTDDATA = 0x1
msac.l	Ry,Rx,<ea>y,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}
msac.w	Ry,Rx,ACCx	PSTDDATA = 0x1
msac.w	Ry,Rx,<ea>y,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}

34.4.5.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. The PSTDDATA specification for these opcodes is shown in Table 34-34.

Table 34-34. PSTDDATA Specification for Supervisor-Mode Instructions

Instruction	Operand Syntax	PSTDDATA Nibble
cpushl	dc,(Ax) ic,(Ax) bc,(Ax)	PSTDDATA = 0x1

halt		PSTDDATA = 0x1,
		PSTDDATA = 0xF

intouch	(Ay)	PSTDDATA = 0x1
move.l	Ay,USP	PSTDDATA = 0x1
move.l	USP,Ax	PSTDDATA = 0x1
move.w	SR,Dx	PSTDDATA = 0x1
move.w	{Dy,#<data>},SR	PSTDDATA = 0x1, {0x3}
movec.l	Ry,Rc	PSTDDATA = 0x1, {8, ASID}

rte		PSTDDATA = 0x7, {0xB, source operand}, {0x3},{ 0xB, source operand}, {DD},
		PSTDDATA = 0x5, {[0x9AB], target address}

stop	#<data>	PSTDDATA = 0x1,
		PSTDDATA = 0xE

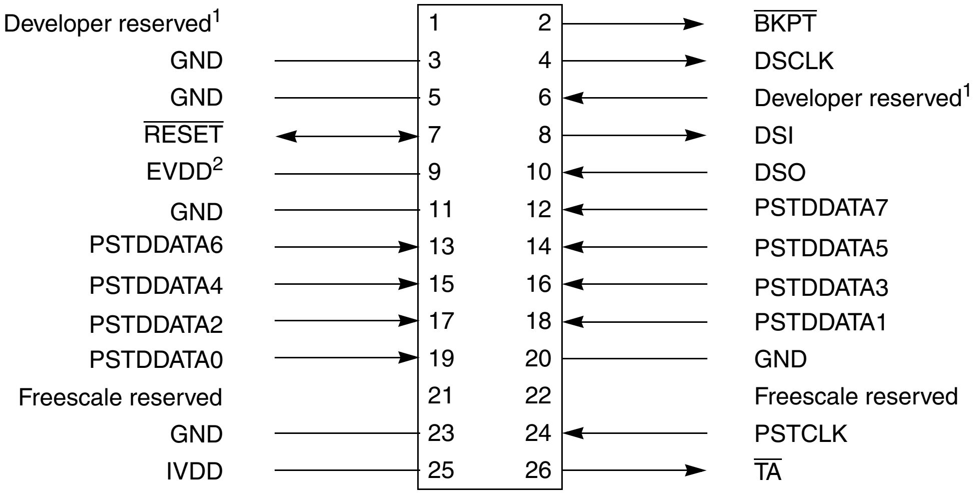
wdebug.l	<ea>y	PSTDDATA = 0x1, {0xB, source, 0xB, source}

The move-to-SR and RTE instructions include an optional PSTDDATA = 0x3 value, indicating an entry into user mode. Additionally, if the execution of a RTE instruction returns the processor to emulator mode, a multiple-cycle status of 0xD is signaled.

Similar to the exception processing mode, the stopped state (PSTDDATA nibble = 0xE) and the halted state (PSTDDATA = 0xFF) display this status throughout the entire time the ColdFire processor is in the given mode.

34.4.6 Freescale-Recommended BDM Pinout

The ColdFire BDM connector is a 26-pin Berg connector arranged 2 x 13 as shown below.



[1] Pins reserved for BDM developer use.
[2] Supplied by target

Figure 34-50. Recommended BDM Connector

Chapter 35
IEEE 1149.1 Test Access Port (JTAG)

35.1 Introduction

The Joint Test Action Group (JTAG) is a dedicated user-accessible test logic compliant with the IEEE 1149.1 standard for boundary-scan testability, which helps with system diagnostic and manufacturing testing.

This architecture provides access to all data and chip control pins from the board-edge connector through the standard four-pin test access port (TAP) and the JTAG reset pin, $\overline{\text{TRST}}$.

35.1.1 Block Diagram

Figure 35-1 shows the block diagram of the JTAG module.

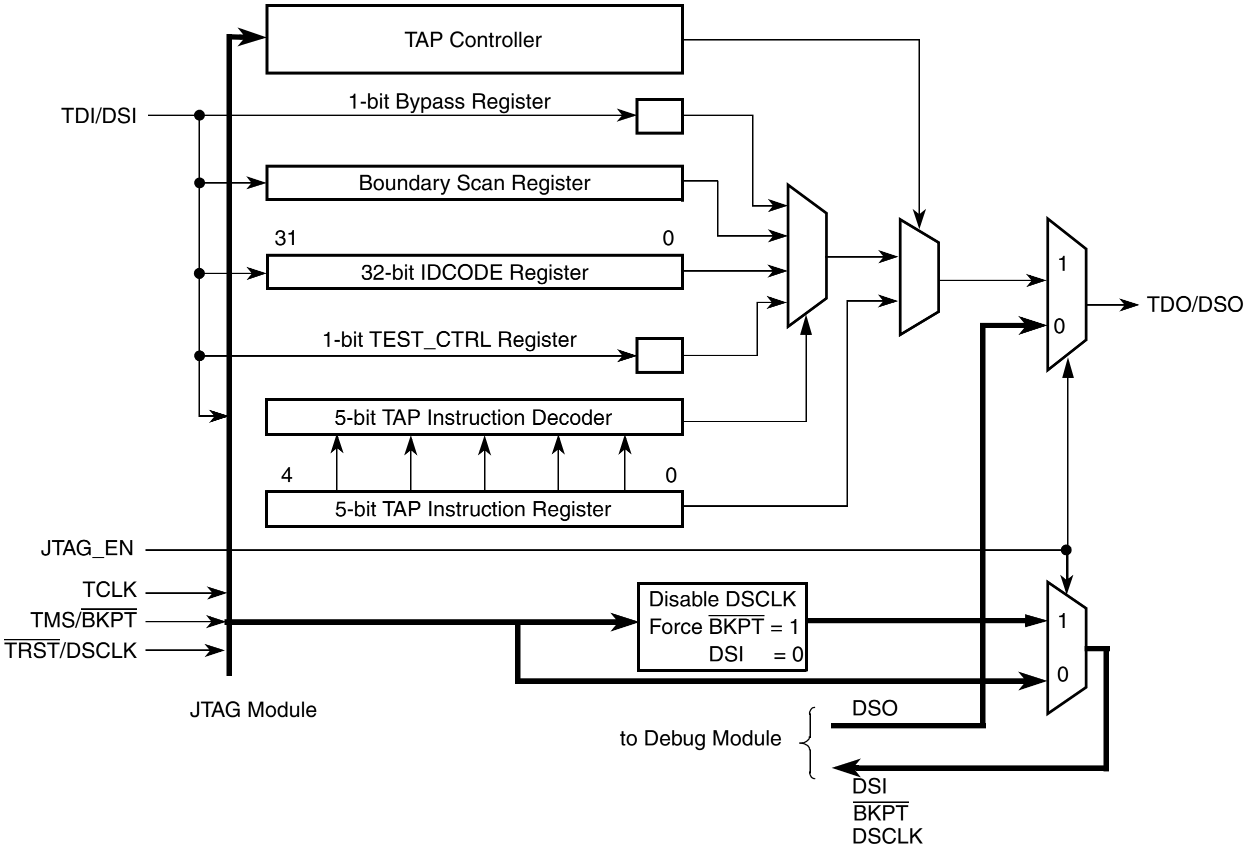


Figure 35-1. JTAG Block Diagram

35.1.2 Features

The basic features of the JTAG module are the following:

- Performs boundary-scan operations to test circuit board electrical continuity
- Bypasses instruction to reduce the shift register path to a single cell
- Sets chip output pins to safety states while executing the bypass instruction
- Samples the system pins during operation and transparently shifts out the result
- Selects between JTAG TAP controller and Background Debug Module (BDM) using a dedicated JTAG_EN pin

35.1.3 Modes of Operation

The JTAG_EN pin can select between the following modes of operation:

- JTAG mode (JTAG_EN = 1)
- Background debug mode (BDM)—for more information, refer to Section 34.4.1, "Background Debug Mode (BDM)"; (JTAG_EN = 0).

35.2 External Signal Description

The JTAG module has five input and one output external signals, as described in Table 35-1.

Table 35-1. Signal Properties

Name	Direction	Function	Reset State	Pull up
JTAG_EN	Input	JTAG/BDM selector input	—	—
TCLK	Input	JTAG Test clock input	—	Active
TMS/$\overline{\text{BKPT}}$	Input	JTAG Test mode select / BDM Breakpoint	—	Active
TDI/DSI	Input	JTAG Test data input / BDM Development serial input	—	Active
$\overline{\text{TRST}}$/DSCLK	Input	JTAG Test reset input / BDM Development serial clock	—	Active
TDO/DSO	Output	JTAG Test data output / BDM Development serial output	Hi-Z / 0	—

35.2.1 JTAG Enable (JTAG_EN)

The JTAG_EN pin selects between the debug module and JTAG. If JTAG_EN is low, the debug module is selected; if it is high, the JTAG is selected. Table 35-2 summarizes the pin function selected depending on JTAG_EN logic state.

Table 35-2. Pin Function Selected

	JTAG_EN = 0	JTAG_EN = 1	Pin Name
Module selected	BDM	JTAG	—
Pin Function	— $\overline{\text{BKPT}}$ DSI DSO DSCLK	TCLK TMS TDI TDO $\overline{\text{TRST}}$	TCLK $\overline{\text{BKPT}}$ DSI DSO DSCLK

When one module is selected, the inputs into the other module are disabled or forced to a known logic level, as shown in Table 35-3, to disable the corresponding module.

Table 35-3. Signal State to the Disable Module

	JTAG_EN = 0	JTAG_EN = 1
Disabling JTAG	$\overline{\text{TRST}}$ = 0 TMS = 1	—
Disabling BDM	—	Disable DSCLK DSI = 0 $\overline{\text{BKPT}}$ = 1

NOTE

The JTAG_EN does not support dynamic switching between JTAG and BDM modes.

35.2.2 Test Clock Input (TCLK)

The TCLK pin is a dedicated JTAG clock input to synchronize the test logic. Pulses on TCLK shift data and instructions into the TDI pin on the rising edge and out of the TDO pin on the falling edge. TCLK is independent of the processor clock. The TCLK pin has an internal pull-up resistor, and holding TCLK high or low for an indefinite period does not cause JTAG test logic to lose state information.

35.2.3 Test Mode Select/Breakpoint (TMS/$\overline{\text{BKPT}}$)

The TMS pin is the test mode select input that sequences the TAP state machine. TMS is sampled on the rising edge of TCLK. The TMS pin has an internal pull-up resistor.

The $\overline{\text{BKPT}}$ pin is used to request an external breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes.

35.2.4 Test Data Input/Development Serial Input (TDI/DSI)

The TDI pin receives serial test and data, which is sampled on the rising edge of TCLK. Register values are shifted in least significant bit (lsb) first. The TDI pin has an internal pull-up resistor.

The DSI pin provides data input for the debug module serial communication port.

35.2.5 Test Reset/Development Serial Clock ($\overline{\text{TRST}}$/DSCLK)

The $\overline{\text{TRST}}$ pin is an active low asynchronous reset input with an internal pull-up resistor that forces the TAP controller to the test-logic-reset state.

The DSCLK pin clocks the serial communication port to the debug module. Maximum frequency is 1/5 the processor clock speed. At the rising edge of DSCLK, data input on DSI is sampled and DSO changes state.

35.2.6 Test Data Output/Development Serial Output (TDO/DSO)

The TDO pin is the lsb-first data output. Data is clocked out of TDO on the falling edge of TCLK. TDO is tri-stateable and actively driven in the shift-IR and shift-DR controller states.

The DSO pin provides serial output data in BDM mode.

35.3 Memory Map/Register Definition

The JTAG module registers are not memory mapped and are only accessible through the TDO/DSO pin.

35.3.1 Instruction Shift Register (IR)

The JTAG module uses a 5-bit shift register with no parity. The IR transfers its value to a parallel hold register and applies an instruction on the falling edge of TCLK when the TAP state machine is in the update-IR state. To load an instruction into the shift portion of the IR, place the serial data on the TDI pin before each rising edge of TCLK. The msb of the IR is the bit closest to the TDI pin, and the lsb is the bit closest to the TDO pin. See Section 35.4.3, “JTAG Instructions” for a list of possible instruction codes.

TAP state: Update-IR Access: User read/write

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset	0	0	0	0	1

Figure 35-2. 5-Bit Instruction Register (IR)

35.3.2 IDCODE Register

The IDCODE is a read-only register; its value is chip dependent. For more information, see Section 35.4.3.1, “IDCODE Instruction”.

IR[4:0]: 0_0001 (IDCODE)

Access: User 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
R	PRN				DC						PIN										JEDEC											ID
W																																
Reset	See note[1]				See note[2]						See note[1]										0	0	0	0	0	0	0	1	1	1	0	1

[1] The reset values for PRN and PIN are device-dependent.

[2] Varies, depending on design center location.

Figure 35-3. IDCODE Register

Table 35-4. IDCODE Field Descriptions

Field	Description
31–28 PRN	Part revision number. Indicate the revision number of the device.
27–22 DC	Freescale design center number.
21–12 PIN	Part identification number. Indicate the device number. 0x04F MCF54450 0x04D MCF54451 0x04B MCF54452 0x049 MCF54453 0x04A MCF54454 0x048 MCF54455
11–1 JEDEC	Joint Electron Device Engineering Council ID bits. Indicate the reduced JEDEC ID for Freescale (0x0E).
0 ID	IDCODE register ID. This bit is set to 1 to identify the register as the IDCODE register and not the bypass register according to the IEEE standard 1149.1.

35.3.3 Bypass Register

The bypass register is a single-bit shift register path from TDI to TDO when the BYPASS, CLAMP, or HIGHZ instructions are selected. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

35.3.4 TEST_CTRL Register

The TEST_CTRL register is a 1-bit shift register path from TDI to TDO when the ENABLE_TEST_CTRL instruction is selected. The TEST_CTRL transfers its value to a parallel hold register on the rising edge of TCLK when the TAP state machine is in the update-DR state. The DSE bit selects the drive strength used in JTAG mode.

IR[4:0]: 0_0110 Access: User read-only

	0
R	DSE
W	
Reset	0

Figure 35-4. 1-Bit TEST_CTRL Register

35.3.5 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST or SAMPLE/PRELOAD instruction is selected. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins or high impedance for tri-stated pins.

The boundary scan register contains bits for bonded-out and non bonded-out signals, excluding JTAG signals, analog signals, power supplies, compliance enable pins, device configuration pins, and clock signals.

35.4 Functional Description

35.4.1 JTAG Module

The JTAG module consists of a TAP controller state machine, which is responsible for generating all control signals that execute the JTAG instructions and read/write data registers.

35.4.2 TAP Controller

The TAP controller is a state machine that changes state based on the sequence of logical values on the TMS pin. Figure 35-5 shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCLK signal.

Asserting the $\overline{\mathrm{TRST}}$ signal asynchronously resets the TAP controller to the test-logic-reset state. As Figure 35-5 shows, holding TMS at logic 1 while clocking TCLK through at least five rising edges also causes the state machine to enter the test-logic-reset state, whatever the initial state.

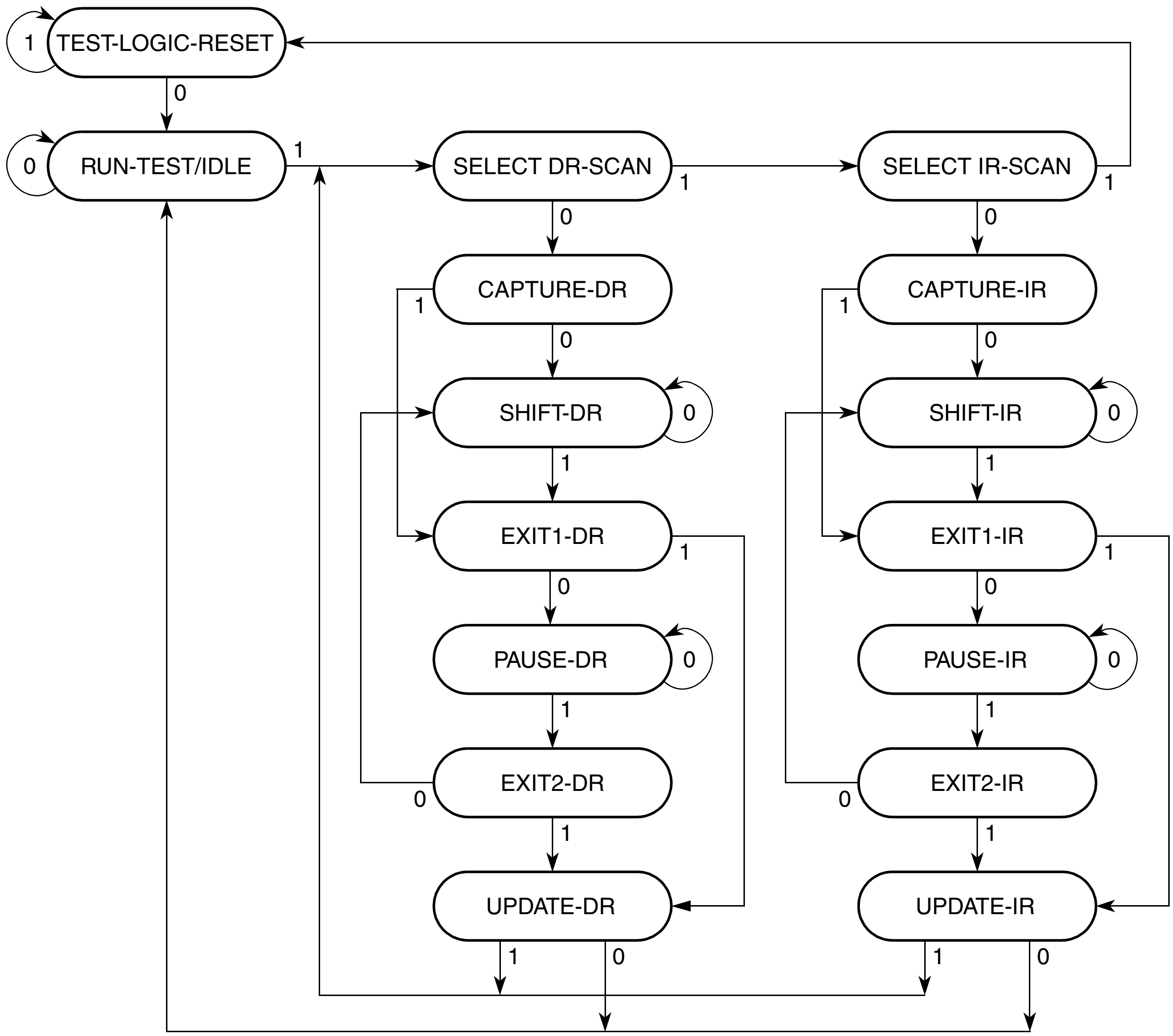


Figure 35-5. TAP Controller State Machine Flow

35.4.3 JTAG Instructions

Table 35-5 describes public and private instructions.

Table 35-5. JTAG Instructions

Instruction	IR[4:0]	Instruction Summary
IDCODE	00001	Selects IDCODE register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation

Table 35-5. JTAG Instructions (continued)

Instruction	IR[4:0]	Instruction Summary
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
ENABLE_TEST_CTRL	00110	Selects TEST_CTRL register
HIGHZ	01001	Selects bypass register while tri-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass while applying fixed values to output pins and asserting functional reset
BYPASS	11111	Selects bypass register for data operations
Reserved	all others[1]	Decoded to select bypass register

[1] Freescale reserves the right to change the decoding of the unused opcodes in the future.

35.4.3.1 IDCODE Instruction

The IDCODE instruction selects the 32-bit IDCODE register for connection as a shift path between the TDI and TDO pin. This instruction allows interrogation of the MCU to determine its version number and other part identification data. The shift register lsb is forced to logic 1 on the rising edge of TCLK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are also forced to fixed values on the rising edge of TCLK following entry into the capture-DR state.

IDCODE is the default instruction placed into the instruction register when the TAP resets. Thus, after a TAP reset, the IDCODE register is selected automatically.

35.4.3.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- SAMPLE - See Section 35.4.3.3, “SAMPLE Instruction,” for description of this function.
- PRELOAD - initialize the boundary scan register update cells before selecting EXTEST or CLAMP. This is achieved by ignoring the data shifting out on the TDO pin and shifting in initialization data. The update-DR state and the falling edge of TCLK can then transfer this data to the update cells. The data is applied to the external output pins by the EXTEST or CLAMP instruction.

35.4.3.3 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and before the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCLK in the capture-DR state when the IR contains the 0x2 opcode. The sampled data is accessible by shifting it through the boundary scan register to the TDO output by using the shift-DR state. The data capture and the shift operation are transparent to system operation.

NOTE

> External synchronization is required to achieve meaningful results because there is no internal synchronization between TCLK and the system clock.

35.4.3.4 EXTEST Instruction

The external test (EXTEST) instruction selects the boundary scan register. It forces all output pins and bidirectional pins configured as outputs to the values preloaded with the SAMPLE/PRELOAD instruction and held in the boundary scan update registers. EXTEST can also configure the direction of bidirectional pins and establish high-impedance states on some pins. EXTEST asserts internal reset for the MCU system logic to force a predictable internal state while performing external boundary scan operations.

35.4.3.5 ENABLE_TEST_CTRL Instruction

The ENABLE_TEST_CTRL instruction selects a 1-bit shift register (TEST_CTRL) for connection as a shift path between the TDI and TDO pin. When the user transitions the TAP controller to the UPDATE_DR state, the register transfers its value to a parallel hold register.

35.4.3.6 HIGHZ Instruction

The HIGHZ instruction eliminates the need to backdrive the output pins during circuit-board testing. HIGHZ turns off all output drivers, including the 2-state drivers, and selects the bypass register. HIGHZ also asserts internal reset for the MCU system logic to force a predictable internal state.

35.4.3.7 CLAMP Instruction

The CLAMP instruction selects the 1-bit bypass register and asserts internal reset while simultaneously forcing all output pins and bidirectional pins configured as outputs to the fixed values that are preloaded and held in the boundary scan update register. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register.

35.4.3.8 BYPASS Instruction

The BYPASS instruction selects the bypass register, creating a single-bit shift register path from the TDI pin to the TDO pin. BYPASS enhances test efficiency by reducing the overall shift path when a device other than the ColdFire processor is the device under test on a board design with multiple chips on the overall boundary scan chain. The shift register lsb is forced to logic 0 on the rising edge of TCLK after entry into the capture-DR state. Therefore, the first bit shifted out after selecting the bypass register is always logic 0. This differentiates parts that support an IDCODE register from parts that support only the bypass register.

35.5 Initialization/Application Information

35.5.1 Restrictions

The test logic is a static logic design, and TCLK can be stopped in a high or low state without loss of data. However, the system clock is not synchronized to TCLK internally. Any mixed operation using the test logic and system functional logic requires external synchronization.

Using the EXTEST instruction requires a circuit-board test environment that avoids device-destructive configurations in which MCU output drivers are enabled into actively driven networks.

Low-power stop mode considerations:

- The TAP controller must be in the test-logic-reset state to enter or remain in the low-power stop mode. Leaving the test-logic-reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
- The TCLK input is not blocked in low-power stop mode. To consume minimal power, the TCLK input should be externally connected to EV_{DD}.
- The TMS, TDI, and $\overline{\text{TRST}}$ pins include on-chip pull-up resistors. For minimal power consumption in low-power stop mode, these three pins should be connected to EV_{DD} or left unconnected.

35.5.2 Nonscan Chain Operation

Keeping the TAP controller in the test-logic-reset state ensures that the scan chain test logic is transparent to the system logic. It is recommended that TMS, TDI, TCLK, and $\overline{\text{TRST}}$ be pulled up. $\overline{\text{TRST}}$ could be connected to ground. However, because there is a pull-up on $\overline{\text{TRST}}$, some amount of current results. The internal power-on reset input initializes the TAP controller to the test-logic-reset state on power-up without asserting $\overline{\text{TRST}}$.

Appendix A
Revision History

This appendix lists major changes between versions of the MCF54455RM document.

A.1 Changes Between Rev. 2 and Rev. 3

Table A-1. Rev. 2 to Rev. 3 Changes

Chapter	Description
Overview	Added PCI as feature on 256-pin devices in family comparison table. On these devices the PCI_AD bus is limited to 24-bits.
SRAM	Corrected minimum core frequency in Features list from 18.75 MHz to 75 MHz
	Added RAMBAR[D/I] bit.
CCM	Corrected FB_AD[7:5] - Flexbus, PCI, Port Size Mode (256-pin Devices) entry in Parallel Configuration During Reset table to match table in CCR Field Descriptions 256-pin's FBCONFIG field.
	In Serial Configuration During Reset table, changed Pins Affected column for PCI and Flexbus A/D Pin Mode to "PCI_AD[31:0] (360-pin) PCI_AD[23:0] (256-pin)"
SBF	The default clock divisor is 67 when first booting from SPI memory, prior to loading the BLDIV value.
SCM	Changed CWSR section note from "If the CWT is enabled, then any write" to "If the CWT is enabled and has not timed out, any write..."
	Changed core watchdog timer functional description section note from "If the CWT is enabled, then any write" to "If the CWT is enabled and has not timed out, any write..."
	Added "The SCMISR[CFEI] bit flags fault errors independent of the CFIER[ECFEI] setting. Therefore, if CFEI is set prior to setting ECFEI, an interrupt is requested immediately after ECFEI is set." to end of SCMISR section.
	Added "**Note:** This bit reports core faults regardless of the setting of CFIER[ECFEI]. Therefore, if the error interrupt is disabled and a core fault occurs, this bit is set. Then, if the error interrupt is subsequently enabled, an interrupt is immediately requested. To prevent an undesired interrupt, clear the captured error by writing one to CFEI before enabling the interrupt." to end of SCMISR[CFEI] bit description.
Interrupt Controller	Reworded Initialization/Application Info section example steps.
	Removed ICONFIG1 register and added note to this section. Similar to the SLMASK and CLMASK registers, there is only one version of this register located in the INTC0 space.
Edge Port	Added bit 0 for each EPORT register, although this bit may not be used on this particular device.
DMA	Added external signal timing section.

Table A-1. Rev. 2 to Rev. 3 Changes (continued)

Chapter	Description
FlexBus	Added notes regarding FlexBus signals tristating between bus cycles throughout.
	Added footnote to FlexBus Signal Summary table regarding signal directions changing during PCI accesses.
	Added indeterminate cycle at the end of the read cycle for address bus in all timing diagrams.
	Added notes in basic read and basic write sections regarding this indeterminate cycle.
	Corrected second sentence in CSMRn[WP] bit description.
	Reworded first entry in results of address comparison table.
	Rearranged FlexBus operating modes table.
	Clarified first sentence in bus cycle states table, S1 Read entry. Moved second sentence into S2 Read entry.
	Changed last sentence in first paragraph in memory map/register definition section from “Reading unused or reserved locations terminates normally and returns zeros.” to “Do not read unused or reserved locations.”
	Added notes in a few sections regarding the number of chip selects depends on the device and its pin configuration
	Added note “When the MCF54450 and MCF54451 devices operate in non-multiplexed mode, only a 24-bit external address is available, FB_A[23:0]” to beginning of chapter.
SDRAM Controller	Added Read Clock Recovery (RCR) Block section.
	Updated SD_DQS signal descriptions.
USB OTG	Changed ID reset value from 0x0041_FA05 to 0x0042_FA05
	Corrected address offsets for the following registers: CAPLENGTH from 0x0100 to 0x0103 HCIVERION from 0x0102 to 0x0100 DCIVERSION from 0x0120 to 0x0122
	Corrected cross-reference in USBCMD[ATDTW] field description.
	Moved USBCMD[ATDTW] from bit location 12 to bit 14. Bit 12 is reserved.
	Changed reset value of FRINDEX from undefined to 0x0000_0000
	Changed OTGSC[1MSS] reset value from 0 to 1.–
	Added “This bit is self clearing,” to EPCR*n*[TXR] and EPCR*n*[RXR] bit descriptions.
	Swapped bit encodings in EPCR*n*[RXI] bit description.
	Changed Total Bytes field to span bits 30–16 instead of 29–16 in Endpoint Transfer Descriptor (dTD) figure.
	Added note to dTD Token[Total Bytes] field.
	Figure USB 2.0 Device States: moved “when the host resets...” arrow from Attach state to Default FS/HS state.
	Added note in Interrupt/Bulk Endpoint Operation section, in third bullet under “RX-dTD is complete when:”
	Added second note to Software Link Pointers section.
	Added the following dTD Token[Total Bytes] field description: “For OUT transfers the total bytes must be evenly divisible by the maximum packet length.”
PCI Controller	Added note at beginning of chapter: “The MCF54450 and MCF54451 devices only contain a 24-bit PCI_AD bus, PCI_AD[23:0].”

Table A-1. Rev. 2 to Rev. 3 Changes (continued)

Chapter	Description
RNG	Added note in overview section.
	Added link to NIST SP800-90 in the overview section.
SSI	Reworded sentences and added tables to register bit descriptions for clarity throughout.
	Changed maximum bit frequency to internal bus clock frequency ratio from 1/4 to 1/5 throughout.
	Changed MSB to msb and LSB to lsb throughout.
RTC	Changed reset value of RTC_DAYS from undefined to 0x0000_0000.
	Added “plus one minute” and note to RTC stopwatch register description.
PIT	Corrected PCSR*n* address in memory map table from 0xFC07_8000 to 0xFC08_8000
	Corrected PIT timeout period equation.
Timers	Clarified DTMR*n*[PS] field description.
DSPI	Corrected first equation in Address Calculation for the First-in and Last-in Entries in the RX FIFO section from “First-in entry address = TX FIFO base + ...” to “First-in entry address = RX FIFO base...”
UART	Reworded note below UART block diagram.
	Corrected note in UIP*n*[CTS] bit description from “...and value as UIPCR*n*[RTS].” to “...and value as UIPCR*n*[CTS].”

A.2 Changes Between Rev. 3 and Rev. 4

Table A-2. Rev. 3 to Rev. 4 Changes

Chapter	Description
Core	Changed reset values for VBR from 0x0000_0000 to undefined for the lower reserved bits.
CCM	The serial configuration depends on the package of the device. Added serial configuration during reset for 256-pin devices table
SBF	256-pin devices use a different SBF reset configuration data. Added new SPI memory organization table for these devices.
Edge Port	Changed EPFR figure’s write row entries to w1c.

Table A-2. Rev. 3 to Rev. 4 Changes (continued)

Chapter	Description
FlexBus	In Results of Address Comparison table corrected result for No CSAR match to "The chip-select signals are not driven. However, the FlexBus runs an external bus cycle with external termination."
	In Overview section corrected maximum FlexBus frequency from 133 to 66MHz.
	In third paragraph of Address and Data Buses, changed "Multiplexed/non-multiplexed operation is determined at reset by FB_AD0." to "Multiplexed/non-multiplexed operation is determined at reset by FB_AD[7:5]."
	In second paragraph of Address and Data Buses, changed from "For example, in 16-bit mode the address continues driving on FB_AD[31:16] and in 8-bit mode the address continues driving on FB_AD[31:8]." to "For example, in 16-bit mode the address continues driving on FB_AD[15:0] and in 8-bit mode the address continues driving on FB_AD[23:0]."
	In FlexBus Multiplexed Operating Modes table, 8- and 16-bit rows, data phase sub-rows, swapped address and data lanes
	In the figure Basic Read-Bus Cycle (No Wait States), updated the bottom signal.
	Removed reset state column from signals description table, since the signals are most likely shared with other functions.
SDRAM Controller	ghtly modified wording in SDCR[DQS_OE] field. Hoping to clarify the fact that each bit of the field operates independently of the other bits.
	Updated Load Mode/Extended Mode Register Command (lmr, lemr) section to clarify some of the information on the SDRAM mode registers and add a description of the mobile DDR extended mode register.
	Edited Initialization/Application Information section to create separate init sequence for each of the supported types of memory.
	Added note to SDCFG1[RD_LAT] field: "**Note:** The recommended values are just a starting point and may need to be adjusted depending on the trace length for the data and DQS lines."
CAU	In the section Introduction, changed "message digest" to "hashing"
FEC	Max buffer size is 2047, not 2032. Changed throughout.
	Reworded EMRBR[R_BUF_SIZE] description.
	In Transmit Buffer Descriptor Field Definitions table, removed the last sentence from the data length field: "Bits [15:5] are used by the DMA engine; bits[4:0] are ignored." to avoid confusion.
Real Time Clock	Corrected RTC_CR[SWR] bit description
DMA Timers	In the table DTMRn Field Descriptions, added the sentence "Avoid setting CLK when RST is set..." to the CLK row description
DSPI	Corrected DSPIn_MCR bit 13 in field description table from DIS_TX to DIS_TXF.
	Corrected DSPIn_MCR bit 11 in field description table from CLR_TX to CLR_TXF.
	Added note to DSPI_MCR[CLR_TXF and CLR_RXF].
	Added note to FIFO Disable Operation section.
I^{2}C	Removed "Support for 3.3-V tolerant devices" from features list as the device supports various other tolerances

Table A-2. Rev. 3 to Rev. 4 Changes (continued)

Chapter	Description
Debug Module	Rearranged sections
	Corrected RTS instruction’s PST definition in PSTDDATA Specification for User-Mode Instructions table
	Corrected “and and” to “&&” in the logical expressions throughout.
	Changed ATTR to AATR in AATR field descriptions
	Changed PBAC address from DRc=0x04 to 0x0A.
	Added note to TDR section: “When performing a level-1, level-2, and level-1 breakpoint sequence, TDR[29] must be cleared in the level-2 breakpoint handler for the second level-1 breakpoint to occur.”
JTAG	Changed reset value of IDCODE[DC] to see note, and added note that this value varies, depending on design center location.

A.3 Changes Between Rev. 4 and Rev. 5

Table A-3. Rev. 4 to Rev. 5 Changes

Chapter	Description
Throughout	Rescinded previous change. The 256-pin devices (MCF54450 and MCF54451) do not contain the PCI bus controller.
FlexBus	Corrected timing diagrams for bursting in multiplexed AD mode. The address on FB_AD does not increment, only the starting address is available during the first bus cycle on FB_AD.
PCI	Changed note at beginning of chapter that the MCF54450 and MCF54451 devices do not contain a PCI bus controller

A.4 Changes Between Rev. 5 and Rev. 6

Table A-4. Rev. 5 to Rev. 6 Changes

Chapter	Description
CCM)	Added part identification number to description of PIN field in Chip Identification Register.
FlexBus	Corrected description of BAM field in CSMRn field description table.
SDRAMC	The "SLE SRE URE AAE SEE FRE PCE UEE UE" bits in the USBINTR register were corrected to read-write. They had been shown as write one to clear.
UART	UART Status Register bit 3 corrected from “TEMP" to “TXEMP" in field description table.
SDRAMC	Made changes to Layout Considerations section because MCF54455 has a true SSTL pad.
Cache	Changed the ACRn register diagram to show the SP bit as write-only.