MCF5329 Reference Manual

Devices Supported:
MCF5327
MCF5328
MCF53281
MCF5329

Document Number: MCF5329RM
Rev. 3
12/2008

freescale™
semiconductor

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 26668334
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

MCF5329RM
Rev. 3
12/2008

Contents

Contents

Chapter 2
Signal Descriptions

Contents

Contents

Chapter 5
Cache

Chapter 6
Static RAM (SRAM)

Chapter 7
Clock Module

Contents

Chapter 8
Power Management

Chapter 9
Chip Configuration Module (CCM)

Contents

Chapter 10
Reset Controller Module

Chapter 11
System Control Module (SCM)

Contents

Chapter 12
Crossbar Switch (XBS)

Chapter 13
General Purpose I/O Module

Contents

Chapter 14
Interrupt Controller Modules

Chapter 15
Edge Port Module (EPORT)

Contents

Chapter 16
Enhanced Direct Memory Access (eDMA)

Contents

Chapter 17
FlexBus

Chapter 18
SDRAM Controller (SDRAMC)

Contents

Chapter 19
Fast Ethernet Controller (FEC)

Contents

Chapter 20
Universal Serial Bus Interface – Host Module

Contents

Chapter 21
Universal Serial Bus Interface – On-The-Go Module

Chapter 22
Liquid Crystal Display Controller (LCDC)

Contents

Contents

Chapter 23
FlexCAN

Chapter 24
Synchronous Serial Interface (SSI)

Contents

Chapter 25
Real-Time Clock

Contents

Chapter 26
Pulse-Width Modulation (PWM) Module

Contents

Chapter 27
Watchdog Timer Module

Chapter 28
Programmable Interrupt Timers (PIT0–PIT3)

Chapter 29
DMA Timers (DTIM0–DTIM3)

Contents

Chapter 30
Queued Serial Peripheral Interface (QSPI)

Contents

Chapter 31
UART Modules

Chapter 32
I^{2}C Interface

Contents

Chapter 33
Message Digest Hardware Accelerator (MDHA)

Contents

Chapter 34
Random Number Generator (RNG)

Chapter 35
Symmetric Key Hardware Accelerator (SKHA)

Contents

Chapter 36
Debug Module

Contents

Chapter 37
IEEE 1149.1 Test Access Port (JTAG)

Appendix A
Register Memory Map Quick Reference

Appendix B
Revision History

About This Book

The primary objective of this reference manual is to define the functionality of the MCF5329 processor for use by software and hardware developers. In addition, this manual supports the MCF5327, MCF5328, and MCF53281. This book is written from the perspective of the MCF5329, and unless otherwise noted, the information applies also to the MCF5327, MCF5328, and MCF53281. The MCF5327, MCF5328, and MCF53281 have the same functionality as the MCF5329 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are in the hardware specifications. Please refer to Table 1-1 to see a summary of the differences.

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, it is the reader's responsibility to be sure he is using 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 20, "Universal Serial Bus Interface – Host Module," and Chapter 21, "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 the MCF5329. 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.

Organization

Following is a summary and brief description of the major sections of this manual:

- Chapter 1, "Overview," includes general descriptions of the modules and features incorporated in the device, focusing in particular on new features.

- Chapter 2, "Signal Descriptions," describes the device signals. It includes a 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.

- Chapter 3, "ColdFire Core," provides an overview of the microprocessor core. The chapter describes the organization of the Version 2 (V2) ColdFire processor core and an overview of the programming model as they are implemented on the device.

- Chapter 4, "Enhanced Multiply-Accumulate Unit (EMAC)," describes the multiply/accumulate unit, which executes integer multiply, multiply-accumulate, and miscellaneous register instructions. The EMAC is integrated into the operand execution pipeline (OEP).

- Chapter 5, "Cache," describes the cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

- Chapter 6, "Static RAM (SRAM)," describes the on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples of how to minimize power consumption when using the SRAM.

- Chapter 7, "Clock Module," describes the device's different clocking methods. It also describes clock module operation in low power modes.

- Chapter 8, "Power Management," describes the low power operation of the device and peripheral behavior in low power modes.

- Chapter 9, "Chip Configuration Module (CCM)," details the various operating configurations of the device. This chapter provides a description of signals used by the CCM and a programming model.

- Chapter 10, "Reset Controller Module," describes the operation of the reset controller module, detailing the different types of reset that can occur.

- Chapter 11, "System Control Module (SCM)," describes the functionality of the SCM, which provides the programming model for peripheral access control, the software core watchdog timer (CWT), and the generic access error information.

- Chapter 12, "Crossbar Switch (XBS)," details the interaction between bus masters and bus slaves within the device, including arbitration schemes.

- Chapter 13, "General Purpose I/O Module," describes the operation and programming model of the general purpose I/O (GPIO) ports on the device.

- Chapter 14, "Interrupt Controller Modules," describes operation of the interrupt controller portion of the SCM. Includes descriptions of the registers in the interrupt controller memory map and the interrupt priority scheme.

- Chapter 15, "Edge Port Module (EPORT)," describes EPORT module functionality, including operation in low power mode.

- Chapter 16, "Enhanced Direct Memory Access (eDMA)," describes the direct memory access (DMA) controller module. It provides an overview of the module and describes in detail its signals and registers. The latter sections of this chapter describe operations, features, and supported data transfer modes in detail.

- Chapter 17, "FlexBus," describes data-transfer operations, chip-select operation, error conditions, bus arbitration, and reset operations.

- Chapter 18, "SDRAM Controller (SDRAMC)," describes the configuration and operation of the SDRAM controller. It begins with a general description 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.

- Chapter 19, "Fast Ethernet Controller (FEC)," provides a feature-set overview, a functional block diagram, and transceiver connection information for MII (media independent interface) and 7-wire serial interfaces. It also provides describes operation and the programming model.

- Chapter 20, "Universal Serial Bus Interface – Host Module," provides an overview of the universal serial bus (USB) host module. The *USB Specification, Revision 2.0* is a recommended supplement to this chapter.

- Chapter 21, "Universal Serial Bus Interface – On-The-Go Module," provides an overview of the universal serial bus (USB) On-the-Go module. The *USB Specification, Revision 2.0* is a recommended supplement to this chapter.

- Chapter 23, "FlexCAN," describes the implementation of the controller area network (CAN) protocol. This chapter describes FlexCAN module operation and provides a programming model.

- Chapter 24, "Synchronous Serial Interface (SSI)," describes SSI module operation and provides a programming model.

- Chapter 25, "Real-Time Clock," describes the real-time clock module operation and provides a programming model.

- Chapter 26, "Pulse-Width Modulation (PWM) Module," describes the configuration and operation of the pulse width modulation (PWM) module. It includes a block diagram, programming model, and functional description.

- Chapter 27, "Watchdog Timer Module," describes software watchdog timer functionality, including operation in low power mode.

- Chapter 28, "Programmable Interrupt Timers (PIT0–PIT3)," describes the functionality of the PIT timers, including operation in low power mode.

- Chapter 29, "DMA Timers (DTIM0–DTIM3)," describes the configuration and operation of the DMA timer modules. These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or triggers. This chapter also provides programming examples.

- Chapter 30, "Queued Serial Peripheral Interface (QSPI)," provides a feature-set overview and a description of operation, including details of the QSPI's internal storage organization. The chapter concludes with the programming model and a timing diagram.

- Chapter 31, "UART Modules," describes the use of the universal asynchronous receiver/transmitters (UARTs) implemented on the device and includes programming examples.

- Chapter 32, "I2C Interface," describes the I^2C module, including I^2C protocol, clock synchronization, and I^2C programming model registers.

- Chapter 33, "Message Digest Hardware Accelerator (MDHA)," describes implementation of two of the world's most popular cryptographic hash functions: SHA-1 and MD5. Accelerators for either algorithm separately have been designed, however the MDHA combines similar functions of the two algorithms into one small, optimized area of silicon on the device.

- Chapter 34, "Random Number Generator (RNG)," describes the 32-bit Random Number Generator (RNG), including a programming model, functional description, and application information.

- Chapter 35, "Symmetric Key Hardware Accelerator (SKHA)," describes the cryptographic hardware coprocessor designed to implement two widely used symmetric key block cipher algorithms, AES and DES.

- Chapter 36, "Debug Module," describes the hardware debug support in the device.

- Chapter 37, "IEEE 1149.1 Test Access Port (JTAG)," describes configuration and operation of the Joint Test Action Group (JTAG) implementation. It describes those items required by the IEEE 1149.1 standard and provides additional information specific to the device. For internal details and sample applications, see the IEEE 1149.1 document.

This manual includes the following appendices:

- Appendix A, "Register Memory Map Quick Reference," provides the entire address map for memory-mapped registers.

- Appendix B, "Revision History," provides a revision history for all previously released versions of this document.

Suggested Reading

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

Hardware Specification

The MCF5329EC document contains the mechanical and electrical specifications of the MCF52329. It can be found at http://www.freescale.com/coldfire.

General Information

The following documentation provides useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual, R1.0* (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.

- Reference manuals (formerly called user's manuals)—These books provide details about individual ColdFire implementations and are intended to be used in conjunction with *The ColdFire Programmers Reference Manual.*
- Addenda/errata to reference manuals—Because some processors have follow-on parts, an addendum is provided that describes the additional features and functionality changes. Also, if mistakes are found within a reference manual, an errata document will be issued before the next published release of the reference manual. These addenda/errata are intended for use with the corresponding reference manuals.
- Hardware specifications—Hardware specifications provide specific data regarding 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 implementation'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
$\overline{\text{OVERBAR}}$	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:
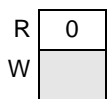
R	0
W	

Indicates a reserved bit field in a memory-mapped register. These bits are always read as zeros.

R	1
W	
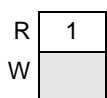
Indicates a reserved bit field in a memory-mapped register. These bits are always read as ones.

---

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.

| 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 | |

Acronyms and Abbreviations

Table 1 lists acronyms and abbreviations used in this document.

Table 1. Acronyms and Abbreviated Terms

Term	Meaning
ADC	Analog-to-digital conversion
ALU	Arithmetic logic unit
BDM	Background debug mode
BIST	Built-in self test
BSDL	Boundary-scan description language
CODEC	Code/decode
DAC	Digital-to-analog conversion
DMA	Direct memory access
DSP	Digital signal processing
EA	Effective address
FIFO	First-in, first-out
GPIO	General-purpose I/O
I^2C	Inter-integrated circuit
IEEE	Institute for Electrical and Electronics Engineers
IFP	Instruction fetch pipeline
IPL	Interrupt priority level
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group

Table 1. Acronyms and Abbreviated Terms (continued)

Term	Meaning
LIFO	Last-in, first-out
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
MAC	Multiply accumulate unit, also Media access controller
MBAR	Memory base address register
MSB	Most-significant byte
msb	Most-significant bit
Mux	Multiplex
NOP	No operation
OEP	Operand execution pipeline
PC	Program counter
PCLK	Processor clock
PLIC	Physical layer interface controller
PLL	Phase-locked loop
POR	Power-on reset
PQFP	Plastic quad flat pack
PWM	Pulse width modulation
QSPI	Queued serial peripheral interface
RISC	Reduced instruction set computing
Rx	Receive
SIM	System integration module
SOF	Start of frame
TAP	Test access port
TTL	Transistor transistor logic
Tx	Transmit
UART	Universal asynchronous/synchronous receiver transmitter
USB	Universal serial bus

Terminology Conventions

Table 2 shows terminology conventions used throughout this document.

Table 2. Notational Conventions

Instruction	Operand Syntax
Opcode Wildcard	
cc	Logical condition (example: NE for not equal)
Register Specifications	
An	Any address register n (example: A3 is address register 3)
Ay,Ax	Source and destination address registers, respectively
Dn	Any data register n (example: D5 is data register 5)
Dy,Dx	Source and destination data registers, respectively
Rc	Any control register (example VBR is the vector base register)
Rm	MAC registers (ACC, MAC, MASK)
Rn	Any address or data register
Rw	Destination register w (used for MAC instructions only)
Ry,Rx	Any source and destination registers, respectively
Xi	Index register i (can be an address or data register: Ai, Di)
Miscellaneous Operands	
#<data>	Immediate data following the 16-bit operation word of the instruction
<ea>	Effective address
<ea>y,<ea>x	Source and destination effective addresses, respectively
<label>	Assembly language program label
<list>	List of registers for MOVEM instruction (example: D3–D0)
<shift>	Shift operation: shift left (<<), shift right (>>)
<size>	Operand data size: byte (B), word (W), longword (L)
bc	Instruction and data caches
dc	Data cache
ic	Instruction cache
# <vector>	Identifies the 4-bit vector number for trap instructions
<>	identifies an indirect data address referencing memory
<xxx>	identifies an absolute address referencing memory
dn	Signal displacement value, n bits wide (example: d16 is a 16-bit displacement)
SF	Scale factor (x1, x2, x4 for indexed addressing mode, <<1n>> for MAC operations)

Table 2. Notational Conventions (continued)

Instruction	Operand Syntax
Operations	
+	Arithmetic addition or postincrement indicator
–	Arithmetic subtraction or predecrement indicator
x	Arithmetic multiplication
/	Arithmetic division
~	Invert; operand is logically complemented
&	Logical AND
|	Logical OR
^	Logical exclusive OR
<<	Shift left (example: D0 << 3 is shift D0 left 3 bits)
>>	Shift right (example: D0 >> 3 is shift D0 right 3 bits)
→	Source operand is moved to destination operand
←→	Two operands are exchanged
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion
If \<condition\> then \<operations\> else \<operations\>	Test the condition. If true, the operations after then are performed. If the condition is false and the optional else clause is present, the operations after else are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.
Subfields and Qualifiers	
{}	Optional operation
()	Identifies an indirect address
d_n	Displacement value, n-bits wide (example: d_{16} is a 16-bit displacement)
Address	Calculated effective address (pointer)
Bit	Bit selection (example: Bit 3 of D0)
lsb	Least significant bit (example: lsb of D0)
LSB	Least significant byte
LSW	Least significant word
msb	Most significant bit
MSB	Most significant byte
MSW	Most significant word

Chapter 1
Overview

The MCF532*x* devices are highly-integrated 32-bit microprocessors based on the Version 3 ColdFire microarchitecture. All MCF532*x* devices contain a 32-Kbyte internal SRAM, an LCD controller, USB host and On-the-Go controllers, a two-bank SDR/DDR SDRAM controller, a 16-channel DMA controller, up to three UARTs, a queued SPI, as well as other peripherals that enable the MCF532*x* family for use in general purpose industrial control applications. Optional peripherals include a fast Ethernet controller, a CAN module, and cryptography hardware accelerators.

This chapter provides an overview of the MCF5327, MCF5328, MCF53281, and MCF5329 microprocessors. It was written from the perspective of the MCF5329 device. See the following section for a summary of differences between the devices.

1.1   MCF532*x* Device Configurations

The following table compares the various devices derivatives available:

Table 1-1. MCF532*x* Family Configurations

Module	MCF5327	MCF5328	MCF53281	MCF5329
ColdFire Version 3 Core with EMAC (Enhanced Multiply-Accumulate Unit)	•	•	•	•
Core (System) Clock	up to 240 MHz			
Peripheral and External Bus Clock (Core clock ÷ 3)	up to 80 MHz			
Performance (Dhrystone/2.1 MIPS)	up to 211			
Unified Cache	16 Kbytes			
Static RAM (SRAM)	32 Kbytes			
LCD Controller	•	•	•	•
SDR/DDR SDRAM Controller	•	•	•	•
USB 2.0 Host	•	•	•	•
USB 2.0 On-the-Go	•	•	•	•
UTMI+ Low Pin Interface (ULPI)	—	•	•	•
Synchronous Serial Interface (SSI)	•	•	•	•
Fast Ethernet Controller (FEC)	—	•	•	•
Cryptography Hardware Accelerators	—	—	—	•
Embedded Voice-over-IP System Solution	—	—	•	—
FlexCAN 2.0B communication module	—	—	•	•

Table 1-1. MCF532*x* Family Configurations (continued)

Module	MCF5327	MCF5328	MCF53281	MCF5329
UARTs	3	3	3	3
I^{2}C	•	•	•	•
QSPI	•	•	•	•
PWM Module	•	•	•	•
Real Time Clock	•	•	•	•
32-bit DMA Timers	4	4	4	4
Watchdog Timer (WDT)	•	•	•	•
Periodic Interrupt Timers (PIT)	4	4	4	4
Edge Port Module (EPORT)	•	•	•	•
Interrupt Controllers (INTC)	2	2	2	2
16-channel Direct Memory Access (DMA)	•	•	•	•
FlexBus External Interface	•	•	•	•
General Purpose I/O Module (GPIO)	•	•	•	•
JTAG - IEEE® 1149.1 Test Access Port	•	•	•	•
Package	196 MAPBGA	256 MAPBGA	256 MAPBGA	256 MAPBGA

1.2   Block Diagram

The superset device in the MCF532*x* family is available in a 256 mold array process ball grid array (MAPBGA) package. Figure 1-1 shows a top-level block diagram of the MCF5329.
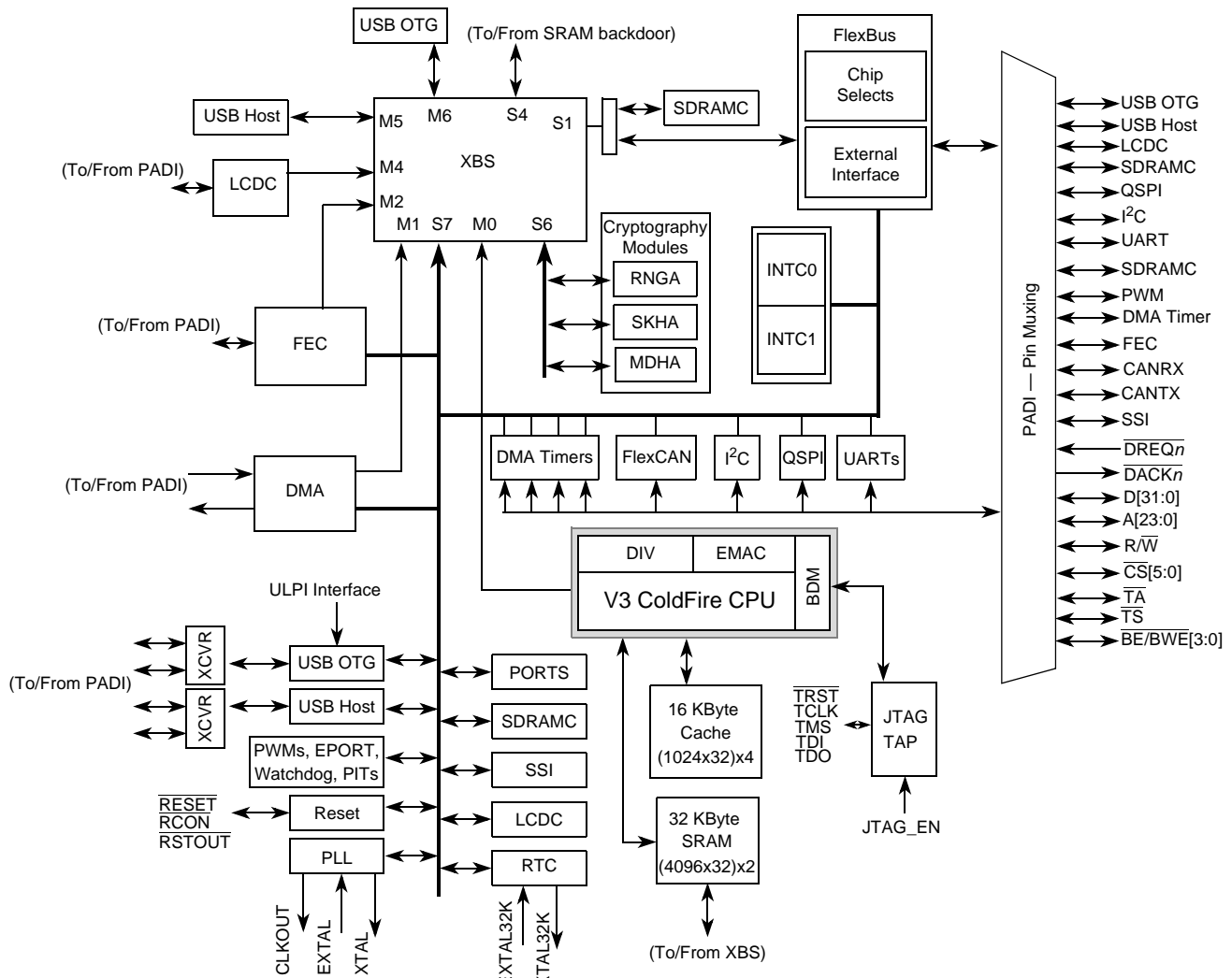
Figure 1-1. MCF5329 Block Diagram

1.3   Features

The following is a brief summary of the functional blocks in the MCF5329 superset device.

- Version 3 ColdFire variable-length RISC processor core
  - Static operation
  - 32-bit address and data path on-chip
  - Processor core runs at three times the bus frequency
  - Sixteen general-purpose 32-bit data and address registers
  - Implements the ColdFire instruction set architecture, ISA_A+, with extensions to support the user stack pointer register, and 4 new instructions for improved bit processing
  - Enhanced multiply-accumulate (EMAC) unit with four 48-bit accumulators to support 32-bit signal processing algorithms

- — Hardware divide execution unit supporting various 32-bit operations
- — Illegal instruction decode that allows for 68K emulation support
- System debug support
  - — Background debug mode (BDM) revision B+ for in-circuit debugging
  - — Real time debug support, with nine user-visible hardware breakpoint registers (PC and address with optional data) that can be configured into a 1- or 2-level trigger
- JTAG support for system level board testing
- On-Chip memories
  - — 16-Kbyte unified write-back cache
  - — 32-Kbyte dual-ported SRAM on CPU internal bus, accessible by core and non-core bus masters (DMA, FEC, LCD controller, and USB host and OTG)
- Power management
  - — Fully static operation with processor wait, doze, and stop modes
  - — Very rapid response to interrupts from sleep mode
  - — Global clock disable register to disable clocks to most modules
  - — Ability to bypass PLL circuitry for low-power and low-speed mode
- Liquid crystal display controller (LCDC)
  - — Support for single (non-split) screen monochrome/color LCD panels and self-refresh type LCD panels
  - — 16 simultaneous gray-scale levels from a palette of 16 for monochrome display
  - — Maximum supported panel size of 800x600 pixels
  - — 4(mapped to RGB444)/8(RGB444)/12 bits per pixel (bpp) for passive color panel
  - — 4(mapped to RGB666)/8(mapped to RGB666)/12(RGB444)/16(RGB565)/18 bpp for TFT
- Embedded voice-over-IP (VoIP) system solution
  - — Fully integrated and tested software VoIP package
- SDR/DDR SDRAM controller
  - — Supports a glueless interface to SDR and DDR SDRAM devices
  - — 16-bit (DDR) or 32-bit (SDR) fixed memory port width
  - — 16 bytes critical word first burst transfer
  - — Up to 14 lines of row address, up to 12 (in 32-bit mode) or 13 (in 16-bit bus mode) column address lines, 2 bits of bank address, and a maximum of two pinned-out chip selects. The maximum row bits plus column bits equals 24 in 32-bit bus mode or 25 in 16-bit mode.
  - — Supports up to 256 MBytes of memory per chip select, 512 MBytes total
  - — Supports page mode to maximize the data rate
  - — Supports sleep and self-refresh modes
- Universal serial bus (USB) host controller
  - — Fully compliant with the *Universal Serial Bus Specification, Revision 2.0*
  - — Support for full speed (FS = 12 Mbps) and low speed (LS = 1.5 Mbps) with on-chip

transceiver in host mode.

— Support for full speed with on-chip transceiver in device mode.

— Compatible with the *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

— Connects to external 5V power control chip for 100mA to 500mA downstream power

— Uses 60 MHz reference clock based off of the system clock or from an external pin

- Universal serial bus (USB) On-the-Go (OTG) controller

— Fully compliant with the *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

— Support for full speed and low speed with on-chip FS/LS transceiver.

— Optional UTMI+ Low Pin Interface (ULPI) available on some packages to support high speed (HS = 480 Mbps) transfers

— Connects to external OTG charge pump and resistor chip via I^2C bus

— Embedded host controller compatible with the *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

— Uses 60 MHz reference clock based on the system clock or from an external pin

- 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, MSB aligned

— Programmable word length up to 24 bits

— AC97 support

- Fast Ethernet controller (FEC)

— 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)

- Cryptography hardware accelerators

— FIPS-140 compliant random number generator

— MD5 and SHA-160 one-way hash algorithms

— DES, Triple-DES, and AES ciphers

- FlexCAN module

— Full implementation of the CAN protocol specification version 2.0B

– Standard Data and Remote Frames (up to 109 bits long)

– Extended Data and Remote Frames (up to 127 bits long)

MCF5329 Reference Manual, Rev 3

    – 0–8 bytes data length

    – Programmable bit rate up to 1 Mbit/sec

  — 16 flexible Message Buffers (MBs) of 0–8 bytes data length each, configurable as Rx or Tx, all supporting standard and extended messages

  — Listen-only mode capability

  — Content-related addressing

  — Three programmable mask registers: global (for MBs 0-13), special for MB14 and special for MB15

  — Programmable transmit-first scheme: lowest ID or lowest buffer number

  — Time stamp based on 16-bit free-running timer

  — Global network time, synchronized by a specific message

- Three universal asynchronous receiver transmitters (UARTs)
  - 16-bit divider for clock generation
  - Interrupt control logic
  - DMA support with separate transmit and receive requests
  - 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
  - Flow control support includes request-to-send ($\overline{\text{U}n\text{RTS}}$) and clear-to-send ($\overline{\text{U}n\text{CTS}}$) lines
- I^{2}C module
  - Interchip bus interface for EEPROMs, 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
- Queued serial peripheral interface (QSPI)
  - Full-duplex, three-wire synchronous transfers
  - Up to three chip selects available
  - Master mode operation only with programmable master bit rates
  - Up to 16 pre-programmed transfers
- Pulse width modulation (PWM) module
  - Four independent PWM channels with programmable period and duty cycle
  - Dedicated counter for each PWM channel
  - Programmable PWM enable/disable for each channel
  - Software selection of PWM duty pulse polarity for each channel
- Real time clock
  - Full clock - days, hours, minutes, seconds
  - Minute countdown timer with interrupt
  - Programmable daily alarm with interrupt

MCF5329 Reference Manual, Rev 3

— Sampling timer with interrupt

— Once-per-day, once-per-hour, once-per-minute, and once-per-second interrupts

— Operation at 32.768 kHz, 32 kHz, or 38.4 kHz (determined by reference clock crystal)

- Four 32-bit DMA timers

  — 12.5-ns resolution at 80 MHz

  — Programmable prescaler and sources for clock input, including an external clock option

  — Input-capture capability with programmable trigger edge on input pin

  — Output-compare with programmable mode for the output pin

  — Free run and restart modes

  — Maskable interrupts and DMA trigger capability on input capture or output compare

- Software watchdog timer

  — 16-bit counter

  — Low-power mode support

- Four periodic interrupt timers (PITs)

  — 16-bit counter

  — Selectable as free running or count down

- Phase locked loop (PLL)

  — 16 MHz reference frequency

  — Programmable dithering

- Interrupt controllers (x2)

  — Support for up to 126 interrupt sources

  — Unique vector number for each interrupt source

  — Ability to mask any individual interrupt source or all interrupt sources (global mask-all)

  — Support for hardware and software interrupt acknowledge (IACK) cycles

  — Combinatorial path to provide wake-up from low power modes

- 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 and destination addresses, transfer size, and 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 four channels

- FlexBus (external interface)

  — Glueless connections to 8-, 16-, or 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 negation, per transfer direction
- — Glueless interface to SRAM devices with or without byte strobe inputs
- — Programmable wait state generator
- — 32-bit bidirectional data bus and 24-bit address bus
- — Up to six chip selects available
- — Byte/write enables (byte strobes)
- — Ability to boot from external memories that are 8, 16, or 32 bits wide
- Chip configuration module (CCM)
  - — System configuration during reset
  - — Unique part identification number and part revision number
- Reset controller
  - – Separate reset in and reset out signals
  - – Five reset sources: power-on reset (POR), external, software, watchdog, PLL loss of lock
  - – Status flag indication of source of last reset
- General purpose I/O interface
  - — Up to 94 bits of GPIO for the MCF5328, MCF53281, and MCF5329
  - — Up to 64 bits of GPIO for the MCF5327
  - — Bit manipulation supported via set/clear functions
  - — Unused peripheral pins may be used as extra GPIO
  - — Programmable drive strength or slew rate control for related group of pins

1.3.1 V3 Core Overview

The Version 3 ColdFire processor core consists of two independent pipeline structures decoupled by an instruction buffer. The four-stage instruction fetch pipeline (IFP) is responsible for instruction-address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions awaiting execution in the Operand Execution Pipeline (OEP). The OEP includes two pipeline stages. The first stage decodes instructions and selects operands (DSOC); the second stage (AGEX) performs instruction execution and calculates operand effective addresses, if needed.

The V3 core implements the ColdFire Instruction Set Architecture Revision A+ with added support for a separate user stack pointer register and four new instructions to assist in bit processing. Additionally, the core includes the enhanced multiply-accumulate unit (EMAC) for improved signal processing capabilities. The EMAC implements a 4-stage execution pipeline, optimized for 32 x 32 bit operations, with support for four 48-bit accumulators. Supported operands include 16- and 32-bit signed and unsigned integers and signed fractional operands, as well as a complete set of instructions to process these data types. The EMAC provides superb support for execution of DSP operations within the context of a single processor at a minimal hardware cost.

The core also includes a hardware divide unit which performs a number of integer-divide operations. The supported divide functions include: 32-bit dividend and 16-bit divisor producing a 16-bit quotient and a

16-bit remainder, 32-bit dividend and 32-bit divisor producing a 32-bit quotient, and 32-bit dividend and 32-bit divisor producing a 32-bit remainder.

1.3.2 Debug Module

The ColdFire processor core debug interface is provided to support system debugging with low-cost debug and emulator development tools. Through a standard debug interface, you can access debug information. This allows the processor and system to be debugged without the need for costly in-circuit emulators.

The on-chip breakpoint resources include a total of nine programmable registers—a pair of upper and lower address registers, a pair of data registers (a 32-bit data register and a 32-bit data mask register), and four 32-bit PC registers plus a 32-bit PC mask register. These registers can be accessed through the dedicated debug serial communication channel or from the processor's supervisor mode programming model. The breakpoint registers can be configured to generate triggers by combining the address, data, and PC conditions in a variety of single or dual-level definitions. The trigger event can be programmed to generate a processor halt or initiate a debug interrupt exception.

To support program trace, the V3 Coldfire core's debug module provides processor status (PST[3:0]) and debug data (DDATA[3:0]) ports. These buses and the PSTCLK output provide execution status, captured operand data, and branch target addresses defining processor activity at one-half the CPU's clock rate.

1.3.3 JTAG

The device supports circuit board test strategies based on the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The test logic includes a test access port (TAP) consisting of a 16-state controller, an instruction register, and three test registers (a bypass register, a boundary-scan register, and an ID register). The boundary scan register links the device's pins into one shift register. Test logic, implemented using static logic design, is independent of the device system logic.

The implementation can do the following:

- Perform boundary-scan operations to test circuit board electrical continuity
- Sample device system pins during operation and transparently shift out the result in the boundary scan register
- Bypass the device for a given circuit board test by effectively reducing the boundary-scan register to a single bit
- Disable the output drive to pins during circuit-board testing
- Drive output pins to stable levels

1.3.4 On-chip Memories

1.3.4.1 Cache

The MCF5329 architecture includes a 16-Kbyte unified cache. This four-way, set-associative cache provides pipelined, single-cycle access on cached instructions and operands.

As with all ColdFire caches, the cache controller implements a non-lockup, streaming design. The use of processor-local memories decouples performance from external memory speeds and increases available bandwidth for external devices or the on-chip DMA module.

The cache implements line-fill buffers to optimize 16-byte line burst accesses. Additionally, the cache supports copyback, write-through, or cache-inhibited modes. A 4-entry, 32-bit buffer is used for cache line push operations and can be configured for deferred write buffering in write-through or cache-inhibited modes.

1.3.4.2   SRAM

The SRAM module provides a general-purpose 32-Kbyte memory block that the ColdFire core can access in a single cycle. 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 quickly service core-initiated accesses or memory-referencing commands from the debug module.

The dual-port SRAM module is also accessible by the DMA, USB host and OTG, LCD, and FEC non-core bus masters through the crossbar switch. The dual-ported nature of the SRAM makes it ideal for implementing applications with double-buffer schemes, where the processor and a bus-mastering device operate in alternate regions of the SRAM to maximize system performance. As an example, system performance can be increased significantly if Ethernet packets are moved from the FEC into the SRAM (rather than external memory) prior to any processing.

1.3.5   LCD Controller

The Liquid Crystal Display Controller (LCDC) provides display data for external gray-scale or color LCD panels. The LCDC is capable of supporting black-and-white, gray-scale, passive-matrix color (passive color or CSTN), and active-matrix color (active color or TFT) LCD panels. The module also provides a direct interface to Sharp 240x320 HR-TFT panels.

1.3.6   Voice-over-IP (VoIP) System Solution

A fully integrated and tested VoIP software development kit is provided with the MCF53281 device. The development kit also includes a royalty-free uClinux™ embedded software BSP complete with source code, GNU tools, kernel, and a broad collection of applications and drivers. This open source BSP is augmented by a complete take-to-market middleware system that includes certified SIP telephony stack, audio subsystem with APIs, and advanced device management system.

The product is suitable for applications that require real-time two-way voice communications and control interfaces, including:

- Building intercom systems
- Building fire and alarm panels
- Emergency panels and poles
- Drive-thru kiosks
- Self-serve kiosks
- Elevator control panels

- Gas bar attendant
- Point-of-sale equipment
- Terminal adapter equipment
- Handsets, spearkerphones
- WiFi phones

Figure 1-2 illustrates the placement of the VoIP development kit in an embedded system solution.
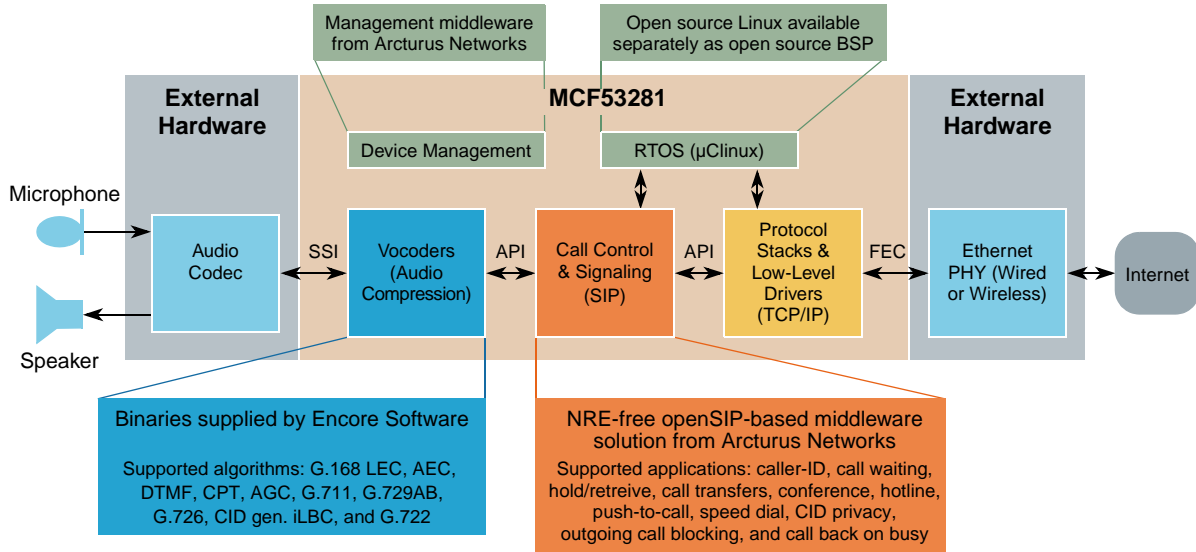
Figure 1-2. MCF53281 Embedded VoIP System Solution

1.3.7 SDR/DDR SDRAM Controller

The SDRAM controller provides a glueless interface to SDR and DDR SDRAM memory devices. The module uses a 32-bit (for SDR) or a 16-bit (for DDR) memory port and can address up to 512 MB of data (256 MB per chip select). The controller supports DDR and SDR SDRAM, but both cannot be used at the same time.

1.3.8 USB Host and OTG Controllers

MCF5329 supports two separate USB 2.0 compliant controllers on chip; a host-only core and an On-The-Go (or dual-role) core. Both controllers support full-speed (12 Mbps) and/or low-speed (1.5 Mbps) USB data rates via on-chip transceivers (The USB OTG module in device mode does not support low-speed). The The USB host module and the USB On-The-Go module's embedded host controllers are compliant with the EHCI driver model and support directly connected full-speed and low-speed devices without the need for UHCI/OHCI companion controllers and associated driver stacks. Both USB controllers contain chaining direct memory access (DMA) engines that reduce interrupt load on the CPU, thereby reducing total system bus bandwidth utilization.

The USB controllers are compliant with the following industry standards:

- *Universal Serial Bus Specification, Revision 2.0*

- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*
- *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*
- *USB 2.0 Transceiver Macrocell Interface (UTMI) Specification, Version 1.05*

1.3.8.1 USB Host Controller

The USB host controller is configured for a single port, which can connect to downstream hubs to support connection of up to 127 devices. The host controller also supports connection to external USB power control devices for downstream power delivery.

1.3.8.2 USB On-the-Go Controller

The second USB controller is programmable to support host, device or On-the-Go operations. The dual-role feature allows device-to-device connectivity, without the need for a host PC (e.g. digital camera to photo printer). The OTG module supports high-speed (480 Mbps) operation via an external ULPI transceiver.

For more details, the following specifications can be found from the ULPI web page at http://www.ulpi.org

- *UTMI+ Specification, Revision 1.0*
- *UTMI+ Low Pin Interface (ULPI) Specification, Revision 1.0*

1.3.9 Synchronous Serial Interface (SSI)

The SSI is a full-duplex, serial port that allows the chip to communicate with a variety of serial devices, including audio codecs, digital signal processors (DSPs), and microprocessors that implement the inter-IC sound bus standard (I^2S) or Intel AC97 standard. SSI typically transfers samples in a periodic manner.

1.3.10 Fast Ethernet Controller (FEC)

The device's integrated fast Ethernet controller (FEC) performs the full set of IEEE® 802.3/Ethernet CSMA/CD media access control and channel interface functions. The FEC supports connection and functionality for the 10/100 Mbps 802.3 media independent interface (MII). It requires an external transceiver (PHY) to complete the interface to the media.

1.3.11 Cryptography Accelerators

The superset device, MCF532973L, incorporates small, fast, dedicated hardware accelerators for random number generation, message digest and hashing, and the DES, 3DES, and AES block cipher functions. This allows for the implementation of common Internet security protocol cryptography operations with performance well in excess of software-only algorithms. Each of the three accelerator modules contains a DMA option for transferring data.

1.3.12 FlexCAN

The FlexCAN module implements the 2.0B CAN protocol that is a commonly used industrial control serial bus that meets the specific requirements of real-time processing, reliable operation in a harsh EMI environment, cost-effectiveness, and required bandwidth. FlexCAN has 16 message buffers.

1.3.13 UARTs

The device contains three independent, full-duplex UARTs. The three UARTs can be clocked by the system bus clock, eliminating the need for an externally supplied clock. They can use DMA requests on transmit-ready and receive-ready as well as interrupt requests for servicing.

1.3.14 I^{2}C Bus

The I^{2}C bus is a two-wire, bidirectional serial bus that provides an efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications requiring occasional communications over a short distance between many devices.

1.3.15 QSPI

The queued serial peripheral interface module provides a high-speed synchronous serial peripheral interface with queued transfer capability. It allows up to 16 transfers to be queued at once, eliminating CPU intervention between transfers.

1.3.16 Pulse Width Modulation (PWM) Timer

The pulse width modulation (PWM) timer generates a synchronous series of pulses having programmable duty cycle. With a suitable low-pass filter, the PWM can be used as a digital-to-analog converter.

The PWM module has four channels, each having a programmable period and duty cycle as well as a dedicated counter. A flexible clock select scheme allows a total of four different clock sources to be used with the counters. Each of the modulators can create independent continuous waveforms with software-selectable duty rates from 0% to 100%. The PWM outputs can be programmed as left-aligned outputs or center-aligned outputs

1.3.17 Real Time Clock

The real time clock module has a dedicated 32/32.768/38.4 kHz crystal oscillator and provides the system with a full clock, capable of interrupting the core once per day, hour, minute, or second. It also contains a sampling timer which can periodically interrupt the core.

1.3.18 DMA Timers (DTIM0-DTIM3)

There are four independent, DMA-transfer-generating 32-bit timers (DTIM[3:0]). Each timer module incorporates a 32-bit timer with a separate register set for configuration and control. The timers can be configured to operate from the system clock or from an external clock source using one of the DT*n*IN

signals. If the system clock is selected, it can be divided by 16 or 1. The input clock is further divided by a user-programmable 8-bit prescaler that clocks the actual timer counter register (TCR*n*). Each of these timers can be configured for input-capture or output-compare mode. By configuring the internal registers, each timer may be configured to assert an external pin, generate an interrupt on a particular event, or cause a DMA transfer.

1.3.19 Software Watchdog Timer

The watchdog timer is a 16-bit timer that facilitates recovery from runaway code. The watchdog counter is a free-running down-counter that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown.

1.3.20 Periodic Interrupt Timers (PIT0–PIT3)

The four periodic interrupt timers (PIT[3:0]) are 16-bit timers that provide precise interrupts at regular intervals with minimal processor intervention. Each timer can count down from the value written in its PIT modulus register, or it can be a free-running down-counter.

1.3.21 Clock Module and Phase Locked Loop (PLL)

The device contains a 16 MHz crystal oscillator, a phase-locked loop, as well as status and control registers. The PLL's output dividers and dithering waveform are register programmable. The system operates via two main clocks generated by the PLL, typically 240 MHz (core) and 80 MHz (peripherals). However, two additional clocks are also generated by the PLL for use by the USB and SDRAM controller modules. To improve noise immunity, the PLL has its own power supply inputs, PLL_VDD and PLL_VSS. All other circuits are powered by the normal internal supply pins, IVDD (core), EVDD (I/O), SD_VDD (SDRAM), and VSS.

The PLL circuitry may be bypassed to reduce system speed and decrease power consumption. The external clock (EXTAL) is used directly, with an optional programmable divider, to produce the internal core and bus clocks.

1.3.22 Interrupt Controllers

There are two interrupt controllers on the MCF5329, which can support up to 126 interrupt sources. Each interrupt source has a unique interrupt vector, and all sources of a given controller provide a programmable level (1-7).

1.3.23 DMA Controller

The implementation of the DMA is targeted towards cost-sensitive applications while providing a high level of functionality. The DMA executes in parallel with the core, enabling transfers of data between the memory and peripherals with little intervention from the core, thus increasing system performance, as well as simplifying software development. The DMA is capable of performing complex data transfers via 16 programmable DMA channels. The hardware microarchitecture includes the DMA engine (which

performs source/destination address calculations and data movement operations), and a dedicated memory array containing transfer control descriptors.

1.3.24 FlexBus External Interface

The FlexBus provides an external interface to 8-, 16-, or 32-bit memory devices (SRAM, flash, ROM, etc.). The FlexBus's internal data lines are shared with the SDRAM controller. When the SDRAMC is in DDR mode (DRAMSEL = 0) the data bus signals, D[31:16], are dedicated to the SDRAM controller and the D[15:0] data bus signals are dedicated to the FlexBus. In SDR mode (DRAMSEL = 1), all 32 data lines are shared between the FlexBus and SDRAM controller.

Features are available to support external flash modules and secondary wait states on reads and writes and a signal to support active-low address valid ($\overline{\text{TS}}$). Six programmable chip-select outputs provide signals to enable external memory and peripheral circuits, providing all handshaking and timing signals for automatic wait-state insertion and data bus sizing.

Base memory address and block size are programmable, with some restrictions. For example, the starting address must be on a boundary that is a multiple of the block size. Each chip select can be configured to provide read and write enable signals suitable for use with most popular static RAMs and peripherals. Data bus width (8-bit, 16-bit, or 32-bit) is programmable on all chip selects, and further decoding is available for protection from user mode access or read-only access.

1.3.25 Reset Controller Module

The reset controller is provided to determine the cause of reset, assert the appropriate reset signals to the system, and keep track of what caused the last reset. There are five sources of reset:

- External
- Power-on reset (POR)
- Watchdog timer
- Phase locked-loop (PLL) loss of lock
- Software

External reset on the $\overline{\text{RSTOUT}}$ pin is software-assertable independent of chip reset state. There are also software-readable status flags indicating the cause of the last reset.

1.3.26 GPIO

Unused bus interface and peripheral pins can be used as discrete general-purpose inputs and outputs. These are managed by a dedicated GPIO module that logically groups all pins into ports located within a contiguous block of memory-mapped control registers. Each port has registers that configure, monitor, and control the port pins. Slew rate control or output pad drive strength control is available on all pins.

Most of the pins associated with the FlexBus interface may be used for several different functions. Their primary function is to provide an external interface to access off-chip resources. When not used for this, the pins may be used as general-purpose digital I/O pins.

1.4 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 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., A23), while designations for multiple signals within a group use brackets (i.e., A[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. Pins that are muxed with GPIO default to their GPIO functionality.

Table 2-1. MCF5327/8/9 Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
Reset								
$\overline{\text{RESET}}$[2]	—	—	—	I	EVDD	J11	N15	N15
$\overline{\text{RSTOUT}}$	—	—	—	O	EVDD	P14	P14	P14

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
Clock								
EXTAL	—	—	—	I	EVDD	L14	P16	P16
XTAL[2]	—	—	—	O	EVDD	K14	N16	N16
EXTAL32K	—	—	—	I	EVDD	M11	P13	P13
XTAL32K	—	—	—	O	EVDD	N11	R13	R13
FB_CLK	—	—	—	O	SDVDD	L1	T2	T2
Mode Selection								
$\overline{\text{RCON}}$[2]	—	—	—	I	EVDD	M7	M8	M8
DRAMSEL	—	—	—	I	EVDD	G11	H12	H12
FlexBus								
A[23:22]	—	$\overline{\text{FB\_CS}}$[5:4]	—	O	SDVDD	B11,C11	C13, D13	C13, D13
A[21:16]	—	—	—	O	SDVDD	B12, A12, D11, C12, B13, A13	E13, A14, B14, C14, A15, B15	E13, A14, B14, C14, A15, B15
A[15:14]	—	SD_BA[1:0][3]	—	O	SDVDD	A14, B14	D14, B16	D14, B16
A[13:11]	—	SD_A[13:11][3]	—	O	SDVDD	C13, C14, D12	C15, C16, D15	C15, C16, D15
A10	—	—	—	O	SDVDD	D13	D16	D16
A[9:0]	—	SD_A[9:0][3]	—	O	SDVDD	D14, E11–14, F11–F14, G14	E14–E16, F13–F16, G16– G14	E14–E16, F13–F16, G16– G14
D[31:16]	—	SD_D[31:16][4]	—	I/O	SDVDD	H3–H1, J4–J1, K1, L4, M2, M3, N1, N2, P1, P2, N3	M1–M4, N1–N4, T3, P4, R4, T4, N5, P5, R5, T5	M1–M4, N1–N4, T3, P4, R4, T4, N5, P5, R5, T5
D[15:1]	—	FB_D[31:17][4]	—	I/O	SDVDD	F4–F1, G5–G2, L5, N4, P4, M5, N5, P5, L6	J3–J1, K4–K1, L2, R6, N7, P7, R7, T7, P8, R8	J3–J1, K4–K1, L2, R6, N7, P7, R7, T7, P8, R8
D0[2]	—	FB_D[16][4]	—	I/O	SDVDD	M6	T8	T8
$\overline{\text{BE}}$/$\overline{\text{BWE}}$[3:0]	PBE[3:0]	$\overline{\text{SD\_DQM}}$[3:0][3]	—	O	SDVDD	H4, P3, G1, M4	L4, P6, L3, N6	L4, P6, L3, N6
$\overline{\text{OE}}$	PBUSCTL3	—	—	O	SDVDD	P6	R9	R9

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
$\overline{TA}$[2]	PBUSCTL2	—	—	I	SDVDD	G13	G13	G13
R/$\overline{W}$	PBUSCTL1	—	—	O	SDVDD	N6	N8	N8
$\overline{TS}$	PBUSCTL0	$\overline{DACK0}$	—	O	SDVDD	D2	H4	H4
Chip Selects								
$\overline{FB\_CS}$[5:4]	PCS[5:4]	—	—	O	SDVDD	—	B13, A13	B13, A13
$\overline{FB\_CS}$[3:1]	PCS[3:1]			O	SDVDD	A11, D10, C10	A12, B12, C12	A12, B12, C12
$\overline{FB\_CS0}$	—	—	—	O	SDVDD	B10	D12	D12
SDRAM Controller								
SD_A10	—	—	—	O	SDVDD	L2	P2	P2
SD_CKE	—	—	—	O	SDVDD	E1	H2	H2
SD_CLK	—	—	—	O	SDVDD	K3	R1	R1
$\overline{SD\_CLK}$	—	—	—	O	SDVDD	K2	R2	R2
$\overline{SD\_CS1}$	—	—	—	O	SDVDD	—	J4	J4
$\overline{SD\_CS0}$	—	—	—	O	SDVDD	E2	H1	H1
SD_DQS3	—	—	—	O	SDVDD	H5	L1	L1
SD_DQS2	—	—	—	O	SDVDD	K6	T6	T6
$\overline{SD\_SCAS}$	—	—	—	O	SDVDD	L3	P3	P3
$\overline{SD\_SRAS}$	—	—	—	O	SDVDD	M1	R3	R3
SD_SDR_DQS	—	—	—	O	SDVDD	K4	P1	P1
$\overline{SD\_WE}$	—	—	—	O	SDVDD	D1	H3	H3
External Interrupts Port[5]								
$\overline{IRQ7}$[2]	PIRQ7[2]	—	—	I	EVDD	J13	J13	J13
$\overline{IRQ6}$[2]	PIRQ6[2]	USBHOST_VBUS_EN	—	I	EVDD	—	J14	J14
$\overline{IRQ5}$[2]	PIRQ5[2]	USBHOST_VBUS_OC	—	I	EVDD	—	J15	J15
$\overline{IRQ4}$[2]	PIRQ4[2]	SSI_MCLK	—	I	EVDD	L13	J16	J16
$\overline{IRQ3}$[2]	PIRQ3[2]	—	—	I	EVDD	M14	K14	K14
$\overline{IRQ2}$[2]	PIRQ2[2]	USB_CLKIN	—	I	EVDD	M13	K15	K15
$\overline{IRQ1}$[2]	PIRQ1[2]	$\overline{DREQ1}$[2]	SSI_CLKIN	I	EVDD	N13	K16	K16

MCF5329 Reference Manual, Rev 3

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
FEC								
FEC_MDC	PFECI2C3	I2C_SCL[2]	—	O	EVDD	—	C1	C1
FEC_MDIO	PFECI2C2	I2C_SDA[2]	—	I/O	EVDD	—	C2	C2
FEC_TXCLK	PFECH7	—	—	I	EVDD	—	A2	A2
FEC_TXEN	PFECH6	—	—	O	EVDD	—	B2	B2
FEC_TXD0	PFECH5	ULPI_DATA0	—	O	EVDD	—	E4	E4
FEC_COL	PFECH4	ULPI_CLK	—	I	EVDD	—	A8	A8
FEC_RXCLK	PFECH3	ULPI_NXT	—	I	EVDD	—	C8	C8
FEC_RXDV	PFECH2	ULPI_STP	—	I	EVDD	—	D8	D8
FEC_RXD0	PFECH1	ULPI_DATA4	—	I	EVDD	—	C6	C6
FEC_CRS	PFECH0	ULPI_DIR	—	I	EVDD	—	B8	B8
FEC_TXD[3:1]	PFECL[7:5]	ULPI_DATA[3:1]	—	O	EVDD	—	D3–D1	D3–D1
FEC_TXER	PFECL4	—	—	O	EVDD	—	B1	B1
FEC_RXD[3:1]	PFECL[3:1]	ULPI_DATA[7:5]	—	I	EVDD	—	E7, A6, B6	E7, A6, B6
FEC_RXER	PFECL0	—	—	I	EVDD	—	D4	D4
LCD Controller								
LCD_D17	PLCDDH1	CANTX	—	O	EVDD	—	—	C9
LCD_D16	PLCDDH0	CANRX	—	O	EVDD	—	—	D9
LCD_D17	PLCDDH1	—	—	O	EVDD	A6	C9	—
LCD_D16	PLCDDH0	—	—	O	EVDD	B6	D9	—
LCD_D15	PLCDDM7	—	—	O	EVDD	C6	A7	A7
LCD_D14	PLCDDM6	—	—	O	EVDD	D6	B7	B7
LCD_D13	PLCDDM5	—	—	O	EVDD	A5	C7	C7
LCD_D12	PLCDDM4	—	—	O	EVDD	B5	D7	D7
LCD_D[11:8]	PLCDDM[3:0]	—	—	O	EVDD	C5, D5, A4, B4	D6, E6, A5, B5	D6, E6, A5, B5
LCD_D7	PLCDDL7	—	—	O	EVDD	C4	C5	C5
LCD_D6	PLCDDL6	—	—	O	EVDD	B3	D5	D5
LCD_D5	PLCDDL5	—	—	O	EVDD	A3	A4	A4
LCD_D4	PLCDDL4	—	—	O	EVDD	A2	A3	A3

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
LCD_D[3:0]	PLCDDL[3:0]	—	—	O	EVDD	D4, C3, D3, B2	B4, C4, B3, C3	B4, C4, B3, C3
LCD_ACD/ LCD_OE	PLCDCTLH0	—	—	O	EVDD	D7	B9	B9
LCD_CLS	PLCDCTLL7	—	—	O	EVDD	C7	A9	A9
LCD_CONTRAST	PLCDCTLL6	—	—	O	EVDD	B7	D10	D10
LCD_FLM/ LCD_VSYNC	PLCDCTLL5	—	—	O	EVDD	A7	C10	C10
LCD_LP/ LCD_HSYNC	PLCDCTLL4	—	—	O	EVDD	A8	B10	B10
LCD_LSCLK	PLCDCTLL3	—	—	O	EVDD	B8	A10	A10
LCD_PS	PLCDCTLL2	—	—	O	EVDD	C8	A11	A11
LCD_REV	PLCDCTLL1	—	—	O	EVDD	D8	B11	B11
LCD_SPL_SPR	PLCDCTLL0	—	—	O	EVDD	B9	C11	C11
USB Host & USB On-the-Go								
USBOTG_M	—	—	—	I/O	USB VDD	G12	L15	L15
USBOTG_P	—	—	—	I/O	USB VDD	H13	L16	L16
USBHOST_M	—	—	—	I/O	USB VDD	K13	M15	M15
USBHOST_P	—	—	—	I/O	USB VDD	J12	M16	M16
FlexCAN (MCF53281 & MCF5329 only)								
CANRX and CANTX do not have dedicated bond pads. Please refer to the following pins for muxing: I2C_SDA, SSI_RXD, or LCD_D16 for CANRX and I2C_SCL, SSI_TXD, or LCD_D17 for CANTX.								
PWM								
PWM7	PPWM7	—	—	I/O	EVDD	—	H13	H13
PWM5	PPWM5	—	—	I/O	EVDD	—	H14	H14
PWM3	PPWM3	DT3OUT	DT3IN	I/O	EVDD	H14	H15	H15
PWM1	PPWM1	DT2OUT	DT2IN	I/O	EVDD	J14	H16	H16
SSI								
SSI_MCLK	PSSI4	—	—	I/O	EVDD	—	G4	G4
SSI_BCLK	PSSI3	$\overline{\text{U2CTS}}$	PWM7	I/O	EVDD	—	F4	F4

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
SSI_FS	PSSI2	$\overline{U2RTS}$	PWM5	I/O	EVDD	—	G3	G3
SSI_RXD[2]	PSSI1	U2RXD	CANRX	I	EVDD	—	—	G2
SSI_TXD[2]	PSSI0	U2TXD	CANTX	O	EVDD	—	—	G1
SSI_RXD[2]	PSSI1	U2RXD	—	I	EVDD	—	G2	—
SSI_TXD[2]	PSSI0	U2TXD	—	O	EVDD	—	G1	—
I²C								
I2C_SCL[2]	PFECI2C1	CANTX	U2TXD	I/O	EVDD	—	—	F3
I2C_SDA[2]	PFECI2C0	CANRX	U2RXD	I/O	EVDD	—	—	F2
I2C_SCL[2]	PFECI2C1	—	U2TXD	I/O	EVDD	E3	F3	—
I2C_SDA[2]	PFECI2C0	—	U2RXD	I/O	EVDD	E4	F2	—
DMA								
$\overline{DACK}$[1:0] and $\overline{DREQ}$[1:0] do not have dedicated bond pads. Please refer to the following pins for muxing: $\overline{TS}$ for $\overline{DACK0}$, DT0IN for $\overline{DREQ0}$, DT1IN for $\overline{DACK1}$, and $\overline{IRQ1}$ for $\overline{DREQ1}$.								
QSPI								
QSPI_CS2	PQSPI5	$\overline{U2RTS}$	—	O	EVDD	P10	T12	T12
QSPI_CS1	PQSPI4	PWM7	USBOTG_PU_EN	O	EVDD	L11	T13	T13
QSPI_CS0	PQSPI3	PWM5	—	O	EVDD	—	P11	P11
QSPI_CLK	PQSPI2	I2C_SCL[2]	—	O	EVDD	N10	R12	R12
QSPI_DIN	PQSPI1	$\overline{U2CTS}$	—	I	EVDD	L10	N12	N12
QSPI_DOUT	PQSPI0	I2C_SDA	—	O	EVDD	M10	P12	P12
UARTs								
$\overline{U1CTS}$	PUARTL7	SSI_BCLK	—	I	EVDD	C9	D11	D11
$\overline{U1RTS}$	PUARTL6	SSI_FS	—	O	EVDD	D9	E10	E10
U1TXD	PUARTL5	SSI_TXD[2]	—	O	EVDD	A9	E11	E11
U1RXD	PUARTL4	SSI_RXD[2]	—	I	EVDD	A10	E12	E12
$\overline{U0CTS}$	PUARTL3	—	—	I	EVDD	P13	R15	R15
$\overline{U0RTS}$	PUARTL2	—	—	O	EVDD	N12	T15	T15
U0TXD	PUARTL1	—	—	O	EVDD	P12	T14	T14
U0RXD	PUARTL0	—	—	I	EVDD	P11	R14	R14
Note: The UART2 signals are multiplexed on the QSPI, SSI, DMA Timers, and I2C pins.								

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
DMA Timers								
DT3IN	PTIMER3	DT3OUT	U2RXD	I	EVDD	C1	F1	F1
DT2IN	PTIMER2	DT2OUT	U2TXD	I	EVDD	B1	E1	E1
DT1IN	PTIMER1	DT1OUT	$\overline{DACK1}$	I	EVDD	A1	E2	E2
DT0IN	PTIMER0	DT0OUT	$\overline{DREQ0}$[2]	I	EVDD	C2	E3	E3
BDM/JTAG[6]								
JTAG_EN[7]	—	—	—	I	EVDD	L12	M13	M13
DSCLK	—	$\overline{TRST}$[2]	—	I	EVDD	N14	P15	P15
PSTCLK	—	TCLK[2]	—	O	EVDD	L7	T9	T9
$\overline{BKPT}$	—	TMS[2]	—	I	EVDD	M12	R16	R16
DSI	—	TDI[2]	—	I	EVDD	K12	N14	N14
DSO	—	TDO	—	O	EVDD	N9	N11	N11
DDATA[3:0]	—	—	—	O	EVDD	N7, P7, L8, M8	N9, P9, N10, P10	N9, P9, N10, P10
PST[3:0]	—	—	—	O	EVDD	N8, P8, L9, M9	R10, T10, R11, T11	R10, T10, R11, T11
Test								
TEST[7]	—	—	—	I	EVDD	E10	A16	A16
PLL_TEST[8]	—	—	—	I	EVDD	—	N13	N13
Power Supplies								
EVDD	—	—	—	—	—	E6, E7, F5–F7, H9, J8, J9, K8, K9, K11	E8, F5–F8, G5, G6, H5, H6, J11, K11, K12, L9–L11, M9, M10	E8, F5–F8, G5, G6, H5, H6, J11, K11, K12, L9–L11, M9, M10
IVDD	—	—	—	—	—	E5, K5, K10, J10	E5, G12, M5, M11, M12	E5, G12, M5, M11, M12
PLL_VDD	—	—	—	—	—	H10	J12	J12
SD_VDD	—	—	—	—	—	E8, E9, F8–F10, J5–J7, K7	E9, F9–F11, G11, H11, J5, J6, K5, K6, L5–L8, M6, M7	E9, F9–F11, G11, H11, J5, J6, K5, K6, L5–L8, M6, M7

Table 2-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
USB_VDD	—	—	—	—	—	G10	L14	L14
VSS	—	—	—	—	—	G6–G9, H6–H8, P9	G7–G10, H7–H10, J7–10, K7–K10, L12, L13	G7–G10, H7–H10, J7–10, K7–K10, L12, L13
PLL_VSS	—	—	—	—	—	H11	K13	K13
USB_VSS	—	—	—	—	—	H12	M14	M14

[1] Refers to pin's primary function.

[2] Pull-up enabled internally on this signal for this mode.

[3] The SDRAM functions of these signals are not programmable by the user. They are dynamically switched by the processor when accessing SDRAM memory space and are included here for completeness.

[4] Primary functionality selected by asserting the DRAMSEL signal (SDR mode). Alternate functionality selected by negating the DRAMSEL signal (DDR mode). The GPIO module is not responsible for assigning these pins.

[5] GPIO functionality is determined by the edge port module. The GPIO module is only responsible for assigning the alternate functions.

[6] If JTAG_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.

[7] Pull-down enabled internally on this signal for this mode.

[8] Must be left floating for proper operation of the PLL.

2.2.1 Internal Pull-up/Pull-downs Resistors

The following table summarizes which external signals contain internal pull-up or pull-down resistors.

Table 2-2. Internal Pull-up/down Resistors

Pin Name	Pull-Up	Pull-Down	Comment
$\overline{\text{RESET}}$	x		Always, except JTAG mode
TEST		x	Always, except JTAG mode
$\overline{\text{RCON}}$	x		Always, except JTAG mode
XTAL	x		When not in crystal oscillator mode (intended for factory test)
$\overline{\text{IRQ}}$[7:2]	x		$\overline{\text{IRQ}}$ mode only
$\overline{\text{IRQ1}}$	x		$\overline{\text{IRQ}}$ and $\overline{\text{DREQ}}$ modes
$\overline{\text{TA}}$	x		Only when used as $\overline{\text{TA}}$
QSPI_DOUT	x		I^2C mode only (I2C_SDA)

Table 2-2. Internal Pull-up/down Resistors (continued)

Pin Name	Pull-Up	Pull-Down	Comment
QSPI_CLK	x		I^{2}C mode only (I2C_SCL)
FEC_MDIO	x		I^{2}C mode only (I2C_SDA)
FEC_MDC	x		I^{2}C mode only (I2C_SCL)
I2C_SDA	x		I^{2}C mode only (I2C_SDA)
I2C_SCL	x		I^{2}C mode only (I2C_SCL)
DT0IN	x		When used as $\overline{DREQ0}$
U1RXD	x	x	When used as SSI_RXD, configured by the MISCCR register in the CCM
U1TXD	x	x	When used as SSI_TXD, configured by the MISCCR register in the CCM
SSI_RXD	x	x	SSI mode only. Configured by the MISCCR register in the CCM
SSI_TXD	x	x	SSI mode only. Configured by the MISCCR register in the CCM
JTAG_EN		x	
TDI	x		JTAG mode only
TMS	x		JTAG mode only
$\overline{TRST}$	x		JTAG mode only
TCLK	x		JTAG mode only
D0	x		During reset only

2.3 Signal Primary Functions

2.3.1 Reset Signals

Table 2-3 describes signals that are used to reset the chip or as a reset indication.

Table 2-3. Reset Signals

Signal Name	Abbreviation	Function	I/O
Reset In	$\overline{RESET}$	Primary reset input to the device. Asserting $\overline{RESET}$ immediately resets the core and peripherals, which stay in reset until $\overline{RESET}$ is negated.	I
Reset Out	$\overline{RSTOUT}$	Reset output ($\overline{RSTOUT}$) is an indicator that the chip is in reset. $\overline{RSTOUT}$ is driven low for 512 FB_CLK clock cycles in response to any internal or external reset.	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 when the internal oscillator circuit is used. The clock source may be configured during reset by asserting $\overline{\text{RCON}}$. See Chapter 9, "Chip Configuration Module (CCM)" for more details.	I
Crystal	XTAL	Used as a connection to the external crystal when the internal oscillator circuit is used to drive the crystal.	O
32 kHz External Clock In	EXTAL32K	32 kHz crystal input clock for the real time clock.	I
32 kHz Crystal	XTAL32K	Oscillator output to EXTAL 32kHz crystal.	O
USB Clock In	USBCLKIN	Allows the user to drive the reference clock to the USB modules, instead of the clock being generated internally by the PLL. This pin should only be driven with a 60 MHz clock. When using the ULPI USB interface, this pin becomes the ULPI input clock.	I
SSI Clock In	SSICLKIN	Allows the user to drive a specific clock frequency to the SSI module, instead of using the internally generated clock.	I
FlexBus Clock Out	FB_CLK	Reflects the internal bus clock (or one-third the core/system clock). ($f_{\text{sys/3}}$)	O

2.3.3 Mode Selection

Table 2-5 describes signals used in mode selection.

Table 2-5. Mode Selection Signals

Signal Name	Abbreviation	Function	I/O
Reset Configuration	$\overline{\text{RCON}}$	Indicates whether the external D[15:0] pin states affect chip configuration at reset.	I
SDR/DDR SDRAM Select	DRAMSEL	Controls whether certain pins act as FlexBus or SDRAMC signals. When asserted, D[31:0] dynamically switches between SDR data and FlexBus data. When negated, D[31:16] are dedicated for DDR data while D[15:0] are dedicated for FlexBus data.	I

2.3.4 FlexBus Signals

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

Table 2-6. FlexBus Signals

Signal Name	Abbreviation	Function	I/O
Address Bus	A[23:0]	The 24 dedicated address signals define the address of external byte, word, and longword accesses. These three-state outputs are the 24 lsbs of the internal 32-bit address bus and multiplexed with the SDRAM controller row and column addresses.	O
Data Bus	D[31:0]	These three-state bidirectional signals provide the general purpose data path between the processor and all other devices.	I/O
Byte Enables	$\overline{BW/BWE}$[3:0]	Define the flow of data on the data bus. During peripheral accesses, these output signals indicate that data is to be latched or driven onto a byte of the data when driven low. The $\overline{BE/BWE}$[3:0] signals are asserted only to the memory bytes used during a read or write access. $\overline{BE/BWE0}$ controls access to the most significant byte lane of data, and $\overline{BE/BWE3}$ controls access to the least significant byte lane of data. For SRAM or Flash devices, the $\overline{BE/BWE}n$ outputs should be connected to individual byte strobe signals. The $\overline{BE/BWE}n$ signals are asserted during accesses to on-chip peripherals, but not to on-chip SRAM or cache. During SDRAM accesses, these signals act as the SD_DQM[3:0] signals, which indicate a byte transfer between SDRAM and the chip when driven high. See Table 2-7 for more details.	O
Output Enable	$\overline{OE}$	Indicates when an external device can drive data during external read cycles.	O
Transfer Acknowledge	$\overline{TA}$	Indicates that the external data transfer is complete. During a read cycle, when the processor recognizes $\overline{TA}$, it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes $\overline{TA}$, the bus cycle is terminated.	I
Read/Write	$R/\overline{W}$	Indicates the direction of the data transfer on the bus for SRAM ($R/\overline{W}$) accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device.	O
Transfer Start	$\overline{TS}$	Bus control output signal indicating the start of a transfer.	O
Chip Selects	$\overline{FB\_CS}$[5:0]	These output signals select external devices for external bus transactions.	O

2.3.5 SDRAM Controller Signals

Table 2-7 describes signals that are used for SDRAM accesses.

Table 2-7. SDRAM Controller Signals

Signal Name	Abbreviation	Function	I/O
SDRAM A10	SD_A10	Bit 10 of the SDRAM Address bus	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 the data bus. SD_DQS1 is tied to SD_DQS3 and SD_DQS0 is tied to SD_DQS2 internally.	I/O
SDR SDRAM Write Data Byte Mask	SD_DQM[3:0]	Used to determine which byte lanes of the data bus should be latched during a write cycle. These pins are multiplexed with the $\overline{\text{BE/BWE}}n$ pins. 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 Synchronous Column Address Strobe	$\overline{\text{SD\_SCAS}}$	SDRAM synchronous column address strobe.	O
SDRAM Synchronous Row Address Strobe	$\overline{\text{SD\_SRAS}}$	SDRAM synchronous row address strobe.	O
SDR SDRAM Data Strobe	SD_SDRDQS	Generated by the memory controller in SDR mode, to mimic the DQS generated by DDR memories during reads. It is routed out and connected back to SD_DQS inputs.	O
SDRAM Write Enable	$\overline{\text{SD\_WE}}$	Indicates the direction of the data transfer on the 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

2.3.6 External Interrupt Signals

Table 2-8 describes the external interrupt signals.

Table 2-8. External Interrupt Signals

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

2.3.7 DMA Signals

Table 2-9 describes the external DMA signals.

Table 2-9. DMA Signals

Signal Name	Abbreviation	Function	I/O
DMA Request	$\overline{\text{DREQ}}$[1:0]	Active low external DMA request lines.	I
DMA Acknowledge	$\overline{\text{DACK}}$[1:0]	Active low external DMA acknowledge lines.	O

2.3.8 LCD Controller Signals

Table 2-9 describes the LCD controller signals.

Table 2-10. LCD Signals

Signal Name	Abbreviation	Function	I/O
Line Data	LCD_D[17:0]	LCD data bus.	O
First Line Marker/ Vertical Sync	LCD_FLM/ LCD_VSYNC	Passive matrix: First line marker Active matrix: Vertical sync pulse. Indicates start of next frame.	O
Line Pulse/ Horizontal Sync	LCD_LP/ LCD_HSYNC	Passive matrix: Line pulse Active matrix: Horizontal sync pulse. Indicates start of next line.	O
Shift Clock	LCD_LSCLK	Clock for latching data into the display driver's internal shift register.	O
Alt. Crystal Direction/ Output Enable	LCD_ACD/ LCD_OE	Passive matrix: Alternate crystal direction Active matrix: Output enable to enable data to be shifted onto the display.	O
Contrast	LCD_CONTRAST	Controls the LCD bias voltage for contrast control.	O
Power Save	LCD_PS	Controls signal output for source driver (Sharp HR-TFT 240x320 panels only)	O
Gate Driver Clock Signal	LCD_CLS	Start signal output for gate driver, inverted version of LCD_PS (Sharp HR-TFT 240x320 panels only).	O
Reverse Control	LCD_REV	Signal for common electrode driving signal preparation (Sharp HR-TFT 240x320 panels only).	O
Sampling Start Signal	LCD_SPL_SPR	Sets the horizontal scan direction (Sharp HR-TFT 240x320 panels only).	O

2.3.9 Ethernet Module (FEC) Signals

The following signals are used by the Ethernet module for data and clock signals.

Table 2-11. Ethernet Module (FEC) Signals

Signal Name	Abbreviation	Function	I/O
Management Data	FEC_MDIO	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC_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_MDC	In Ethernet mode, FEC_MDC is an output clock which provides a timing reference to the PHY for data transfers on the FEC_MDIO signal. Applies to MII mode operation.	O

Table 2-11. Ethernet Module (FEC) Signals (continued)

Signal Name	Abbreviation	Function	I/O
Collision	FEC_COL	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.	I
Carrier Receive Sense	FEC_CRS	When asserted, indicates that transmit or receive medium is not idle. Applies to MII mode operation.	I
Transmit Clock	FEC_TXCLK	Input clock which provides a timing reference for FEC_TXEN, FEC_TXD[3:0] and FEC_TXER	I
Transmit Enable	FEC_TXEN	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.	O
Transmit Data 0	FEC_TXD0	FEC_TXD0 is the serial output Ethernet data and is only valid during the assertion of FEC_TXEN. This signal is used for 10-Mbps Ethernet data. It is also used for MII mode data in conjunction with FEC_TXD[3:1].	O
Transmit Data 1–3	FEC_TXD[3:1]	In Ethernet mode, these pins contain the serial output Ethernet data and are valid only during assertion of FEC_TXEN in MII mode.	O
Transmit Error	FEC_TXER	In Ethernet mode, when FEC_TXER is asserted for one or more clock cycles while FEC_TXEN is also asserted, the PHY sends one or more illegal symbols. FEC_TXER has no effect at 10 Mbps or when FEC_TXEN is negated. Applies to MII mode operation.	O
Receive Clock	FEC_RXCLK	Provides a timing reference for FEC_RXDV, FEC_RXD[3:0], and FEC_RXER.	I
Receive Data Valid	FEC_RXDV	Asserting the FEC_RXDV input indicates that the PHY has valid nibbles present on the MII. FEC_RXDV should 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.	I
Receive Data 0	FEC_RXD0	FEC_RXD0 is the Ethernet input data transferred from the PHY to the media-access controller when FEC_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_RXD[3:1].	I
Receive Data 1–3	FEC_RXD[3:1]	In Ethernet mode, these pins contain the Ethernet input data transferred from the PHY to the media access controller when FEC_RXDV is asserted in MII mode operation.	I
Receive Error	FEC_RXER	In Ethernet mode, FEC_RXER—when asserted with FEC_RXDV—indicates that the PHY has detected an error in the current frame. When FEC_RXDV is not asserted FEC_RXER has no effect. Applies to MII mode operation.	I

2.3.10   I^{2}C I/O Signals

Table 2-12 describes the I^{2}C serial interface module signals.

Table 2-12. I²C I/O Signals

Signal Name	Abbreviation	Function	I/O
Serial Clock	I2C_SCL	Open-drain clock signal for the I²C interface. It is driven by the I²C module when the bus is in the master mode or it becomes the clock input when the I²C is in the slave mode.	I/O
Serial Data	I2C_SDA	Open-drain signal that serves as the data input/output for the I²C interface.	I/O

2.3.11 FlexCAN Signals

Table 2-13 describes the FlexCAN module signals.

Table 2-13. FlexCAN Signals

Signal Name	Abbreviation	Function	I/O
FlexCAN Transmit	CANTX	Controller area network transmit data output.	O
FlexCAN Receive	CANRX	Controller area network receive data input.	I

2.3.12 Queued Serial Peripheral Interface (QSPI)

Table 2-14 describes QSPI signals.

Table 2-14. Queued Serial Peripheral Interface (QSPI) Signals

Signal Name	Abbreviation	Function	I/O
QSPI Syncrhonous Serial Output	QSPI_DOUT	Provides the serial data from the QSPI and can be programmed to be driven on the rising or falling edge of QSPI_CLK. Each byte is sent msb first.	O
QSPI Synchronous Serial Data Input	QSPI_DIN	Provides the serial data to the QSPI and can be programmed to be sampled on the rising or falling edge of QSPI_CLK. Each byte is written to RAM lsb first.	I
QSPI Serial Clock	QSPI_CLK	Provides the serial clock from the QSPI. The polarity and phase of QSPI_CLK are programmable. The output frequency is programmed according to the following formula, in which n can be any value between 1 and 255: $$SPI\_CLK = f_{sys/3} \div (2 \times n)$$	O
Synchronous Peripheral Chip Selects	QSPI_CS[2:0]	Provide QSPI peripheral chip selects that can be programmed to be active high or low.	O

2.3.13 Synchronous Serial Interface (SSI) Signals

The SSI module uses the signals in this section.

Table 2-15. SSI Module Signals

Signal Name	Abbreviation	Function	I/O
Serial Clock	SSI_CLK	Used by the receive and transmit blocks. In gated clock mode, SSI_CLK is only valid during the transmission of data, otherwise it is pulled to an inactive state.	I/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 riding edge of SSI_CLK.	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

2.3.14 Universal Serial Bus (USB) Signals

The following table describes the signals for the USB module.

Table 2-16. USB Module Signals

Signal Name	Abbreviation	Function	I/O
On-the-Go D-	USBOTG_DM	D- output of the dual-speed transceiver for the On-the-Go module.	O
On-the-Go D+	USBOTG_DP	D+ output of the dual-speed transceiver for the On-the-Go module.	O
On-the-Go Enable	USBOTG_PU_EN	Enables an external pull-up on the USBOTG_DP line. This signal is controlled by the UOCSR[BVLD] bit.	O
Host D-	USBHOST_DM	D- output of the dual-speed transceiver for the USB Host module.	O
Host D+	USBHOST_DP	D+ output of the dual-speed transceiver for the USB Host module.	O
Host VBUS Enable	USBHOST_VBUS_EN	Enables off-chip VBUS charge pump	O
Host VBUS over-current	USBHOST_VBUS_OC	Indicates to the processor that a short has occurred on the USB data bus.	I
ULPI Data Bus	ULPI_DATA[7:0]	Data bus for the ULPI interface, which is synchronous to USBCLKIN/ULPI_CLK.	I/O
ULPI Data Bus Direction	ULPI_DIR	Indicates direction of the ULPI data bus., which is synchronous to USBCLKIN/ULPI_CLK	I
ULPI Stop Data	ULPI_STP	Synchronous to USBCLKIN	
ULPI Next Data	ULPI_NXT	Synchronous to USBCLKIN	
ULPI On-Chip Clock	ULPI_CLK	60MHz clock which is generated on-chip	O
USB Off-Chip Clock	USBCLKIN	See Section 2.3.2, "PLL and Clock Signals"	I

2.3.15 Pulse Width Modulation (PWM) Module Signals

The following table describes the signals for the PWM module.

Table 2-17. PWM Module Signals

Signal Name	Abbreviation	Function	I/O
PWM7 Output	PWM7	Waveform output for channel 7 of the PWM module. Also functions as an input for the emergency shutdown feature of the PWM.	I/O
PWM[5,3,1,0] Outputs	PWM[5,3,1,0]	Waveform output for channels 5, 3, 1, and 0 respectively.	O

2.3.16 UART Module Signals

Table 2-18 describes the signals of the three UART modules, where *n*=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	Transmitter serial data outputs. Data is shifted out lsb first on this pin at the falling edge of the serial clock source. The output is held high when the transmitter is disabled, idle, or in local loopback mode.	O
Receive Serial Data Input	U*n*RXD	Receiver serial data inputs. Data is sampled on the rising edge of the serial clock source lsb first. When the UART clock is stopped for power-down mode, any transition on this pin restarts it.	I
Clear-to-Send	$\overline{\text{U}n\text{CTS}}$	Indicates that the UART modules can begin data transmission	I
Request-to-Send	$\overline{\text{U}n\text{RTS}}$	Automatic request-to-send outputs from the UART modules. They may also be asserted and negated as a function of the receive FIFO level.	O

2.3.17 DMA Timer Signals

Table 2-19 describes the signals of the four DMA timer modules, where *n*=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 to occur 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	The output from the 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
Test Reset	$\overline{\text{TRST}}$	This active-low signal is 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 is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O
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	This internally-synchronized signal provides data input for the serial communication port to the BDM module.	I
Development Serial Output	DSO	This internally-registered signal provides serial output communication for BDM module responses.	O
Processor Status Clock	PSTCLK	Used by the development system to known when to sample the DDATA and PST signals.	O
Debug Data	DDATA[3:0]	Display captured processor data and breakpoint status. The PSTCLK signal can be used by the development system to know when to sample DDATA[3:0].	O
Processor Status Outputs	PST[3:0]	Indicate core status, as shown in Table 2-21. Debug mode timing is synchronous with the processor clock; status is unrelated to the current bus transfer. The PSTCLK signal can be used by the development system to know when to sample PST[3:0].	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 DDATA

Table 2-21. Processor Status (continued)

PST[3:0]	Processor Status
1001	Begin two-byte transfer on DDATA
1010	Begin three-byte transfer on DDATA
1011	Begin four-byte transfer on DDATA
1100	Exception processing
1101	Reserved
1110	Processor is stopped
1111	Processor is halted

2.3.19 Test Signals

Table 2-22 describes test signals which are 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 chip. 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	PLL_VDD PLL_VSS	Dedicated power supply signals to isolate the sensitive PLL analog (VCO) 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 controllers.	—
USB Ground	USB_VSS	These pins are the negative supply (ground) for the USB controllers.	—
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 3 (V3) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, see the ISA_A+ definition in the *ColdFire Family Programmer's Reference Manual*. The V3 ColdFire core emphasizes operating frequency and system performance and provides backward object file compatibility to the Version 2 (V2) ColdFire core. It is a step on the ColdFire core roadmap of providing higher performance embedded microprocessors. Specific enhancements include a 4-stage instruction fetch pipeline (IFP) with an 8-entry instruction buffer and change-of-flow acceleration, a 2-stage pipeline local bus structure, and a 4-way set-associative unified cache design supporting copyback and write-through modes of operation.

3.1.1   Overview

As with all ColdFire cores, the V3 ColdFire core is comprised of two separate pipelines decoupled by an instruction buffer.
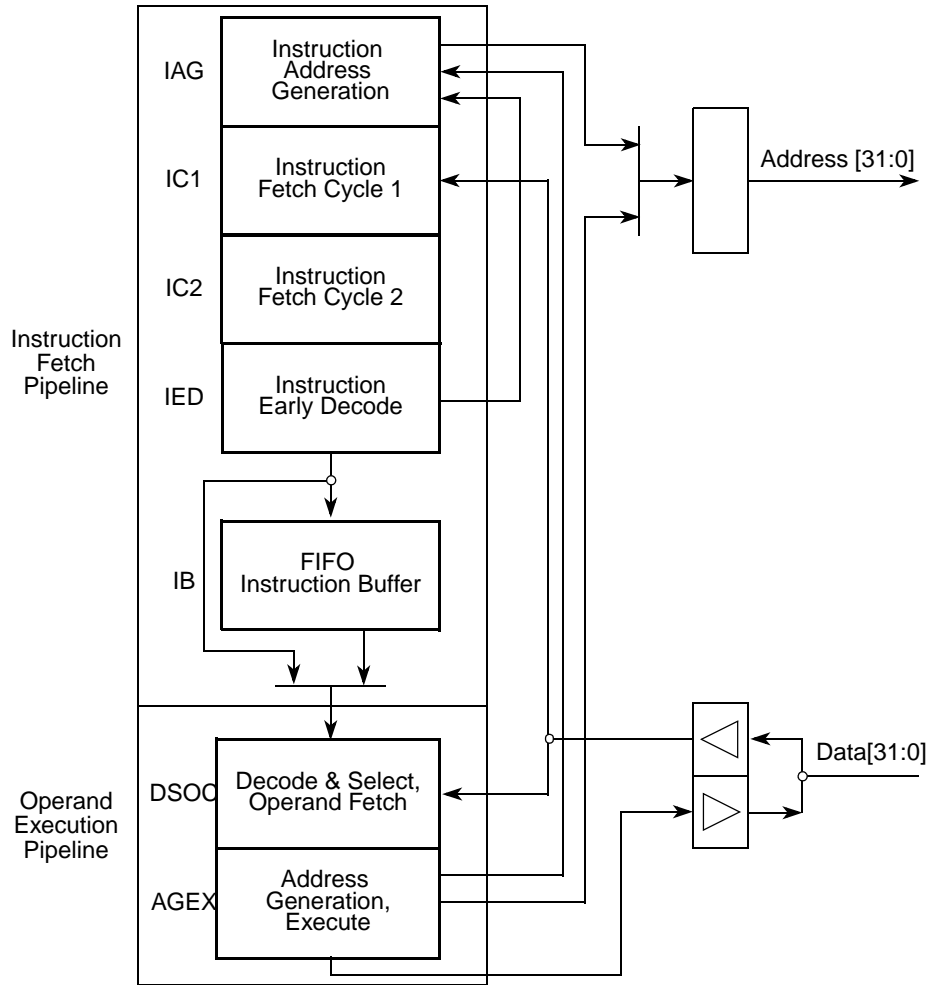
Figure 3-1. V3 ColdFire Core Pipelines

The instruction fetch pipeline (IFP) is a four-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the two-stage operand execution pipeline (OEP), which 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 V3 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

- Two-stage operand execution pipeline (OEP)
  - Decode and select/operand fetch cycle (DSOC)—Decodes instructions and fetches the required components for effective address calculation, or the operand fetch cycle
  - Address generation/execute cycle (AGEX)—Calculates operand address or executes the instruction

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 informration in the IB until it is required by the OEP.

For register-to-register and register-to-memory store operations, the instruction passes through both OEP stages once. For memory-to-register and read-modify-write memory operations, an instruction is effectively staged through the OEP twice: the first time to calculate the effective address and initiate the operand fetch on the processor's local bus, and the second time to complete the operand reference and perform the required function defined by the instruction.

The V3 ColdFire core's instruction buffer is organized differently than the V2 ColdFire core's. One of the time-critical decode fields provided by the early-decode stage of the IFP is the instruction length. By knowing the length of the prefetched instructions, the IED field can package the fetched data into machine instructions and load them into the FIFO instruction buffer in that form. This approach greatly simplifies and accelerates the OEP read logic. As one instruction is completed in the OEP, the next instruction—regardless of instruction length—is read from the next sequential buffer location and loaded into the instruction registers.

The resulting pipeline and local bus structure allow the V3 ColdFire core to deliver sustained high performance across a variety of demanding embedded applications.

3.1.1.1 Change-of-Flow Acceleration

Because the IFP and OEP are decoupled by the instruction buffer, the increased depth of the IFP is generally hidden from the OEP's instruction execution. However, for change-of-flow instructions, such as unconditional branches or jumps, subroutine calls, taken conditional branches, the increased IFP depth is fully exposed. To minimize the effects of this increased depth, a logic module dedicated to change-of-flow acceleration was developed for the IED stage of the IFP.

The basic premise of the V3 ColdFire core's branch acceleration is to detect certain types of change-of-flow instructions, calculate their target instruction address, and immediately begin fetching down the target stream. By allowing the IFP to manage switching of the prefetch stream without OEP intervention, typical execution time is greatly improved.

For example, consider a PC-relative unconditional branch using the BRA instruction. The branch acceleration logic searches the prefetch stream for this type of opcode. After encountered, the acceleration logic calculates the target address by summing the current instruction prefetch address with a displacement contained in the instruction. This detection and calculation of the target address occurs in the IED stage of the BRA prefetch. The target address is then immediately fed back into the IAG stage, causing the current prefetch stream to be discarded and establishing a new stream at the target address. Given that the two pipelines are decoupled, in many cases, the target instruction is available to the OEP immediately after the BRA instruction, making its execution time appear as a single cycle.

The acceleration logic uses a static prediction algorithm when processing conditional branch (Bcc) instructions. The default prediction scheme is as follows: forward Bcc instructions are predicted as not taken, while backward Bcc opcodes are predicted as taken. A user-mode bit in the condition control register, CCR[P], supports altering the prediction dynamically for forward Bcc instructions. See Section 3.2.4, "Condition Code Register (CCR)."

Depending on the run-time characteristics of an application, processor performance may be increased significantly by setting or clearing this configuration bit. Section 3.3.5.7, "Branch Instruction Execution Times," gives details on individual instruction performance.

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 4, "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, and 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, which 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)
- One 32-bit memory base address register (RAMBAR)

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	0xCF30_60	No	3.2.1/3-6
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x0000_0670	No	3.2.1/3-6
Load: 0x082–7 Store: 0x182–7	Data Register 2–7 (D2–D7)	32	R/W	Undefined	No	3.2.1/3-6
Load: 0x088–8E Store: 0x188–8E	Address Register 0–6 (A0–A6)	32	R/W	Undefined	No	3.2.2/3-6
Load: 0x08F Store: 0x18F	Supervisor/User A7 Stack Pointer (A7)	32	R/W	Undefined	No	3.2.3/3-6
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	No	4.2.1/4-3
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	No	4.2.2/4-5
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	No	4.2.3/4-6
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	No	4.2.4/4-7
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	No	4.2.4/4-7
0x80E	Condition Code Register (CCR)	8	R/W	Undefined	No	3.2.4/3-7
0x80F	Program Counter (PC)	32	R/W	Contents of location 0x0000_0004	No	3.2.5/3-8
Supervisor Access Only Registers						
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	3.2.6/3-8
0x004–5	Access Control Register 0–1 (ACR0–1)	32	R/W	See Section	Yes	3.2.7/3-9
0x800	User/Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x0000_0000	No	3.2.3/3-6
0x801	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes	3.2.8/3-9
0x80E	Status Register (SR)	16	R/W	0x27--	No	3.2.9/3-9
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	3.2.10/3-10

[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 36, "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*)   Access: User read/write
Store: 0x180 + *n; n* = 0-7 (D*n*)   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"
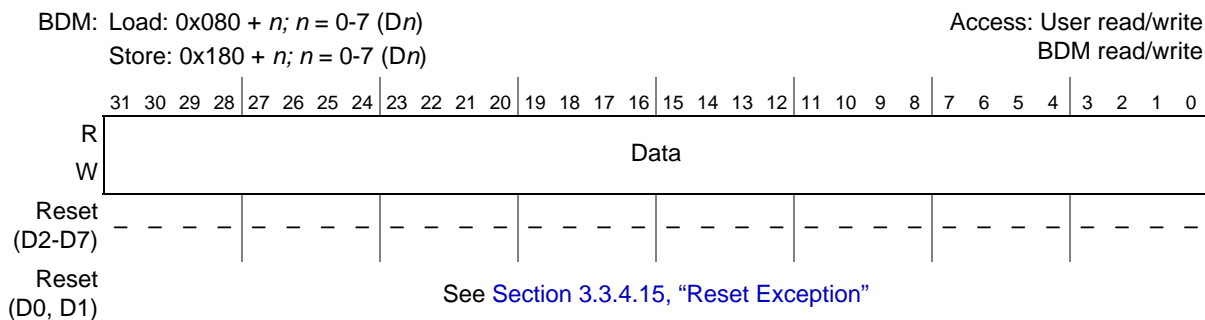
Figure 3-2. Data Registers (D0–D7)

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*)   Access: User read/write
Store: 0x188 + *n; n* = 0–6 (A*n*)   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
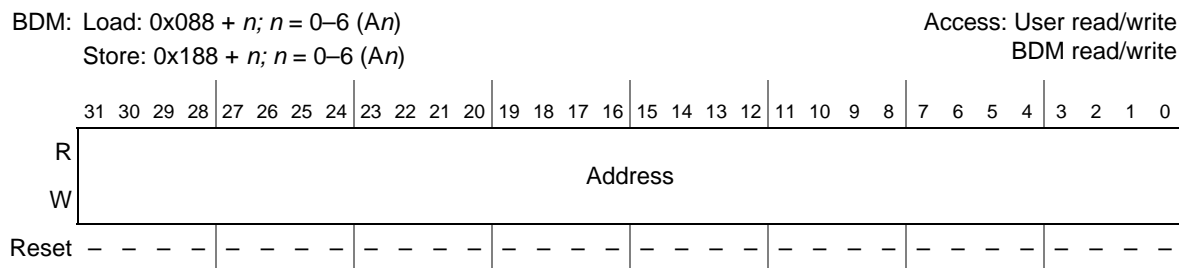
Figure 3-3. Address Registers (A0–A6)

3.2.3   Supervisor/User Stack Pointers (A7 and OTHER_A7)

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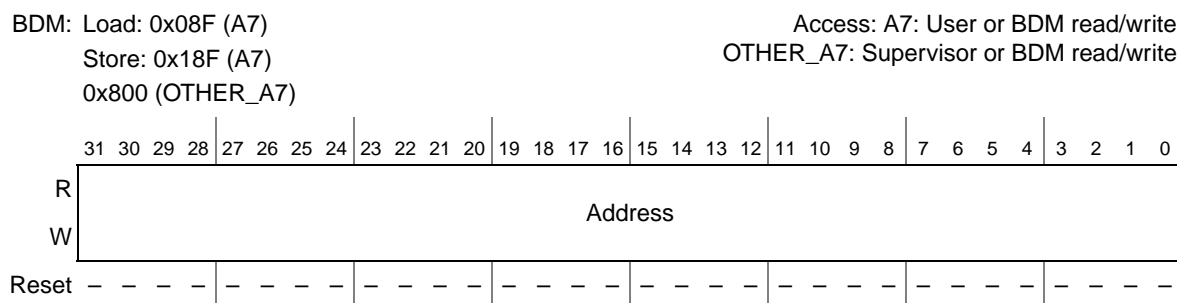
Figure 3-4. Stack Pointer Registers (A7 and OTHER_A7)

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. The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.
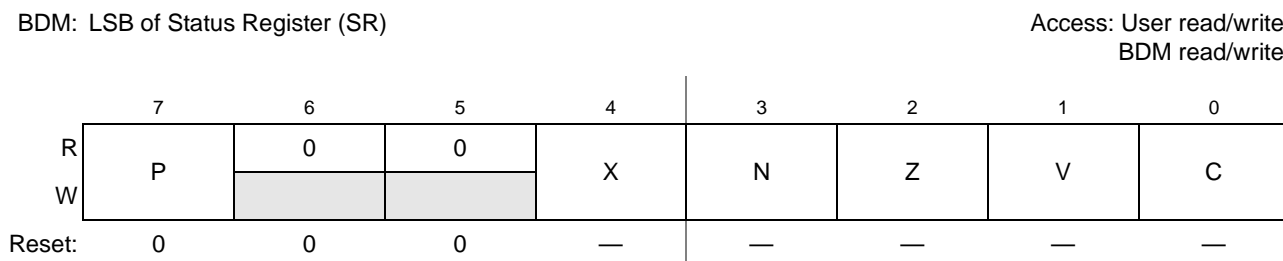
Figure 3-5. Condition Code Register (CCR)

Table 3-2. CCR Field Descriptions

Field	Description
7 P	Branch prediction bit. Alters the static prediction algorithm used by the branch acceleration logic in the IFP on forward conditional branches. 0 Predicited as not taken. 1 Predicted as taken.
6–5	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.
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 contents of the PC or places a new value in the PC, as appropriate. The PC is a base address for PC-relative operand addressing.

The PC is initially loaded during reset exception processing with the contents of location 0x0000_0004.
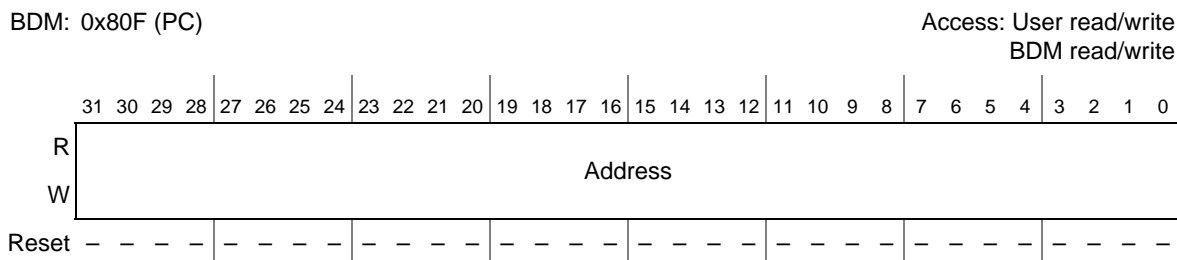
Figure 3-6. Program Counter Register (PC)

3.2.6 Cache Control Register (CACR)

The CACR controls operation of the instruction/data cache memories. It includes bits for enabling, freezing, and invalidating cache contents. It also includes bits for defining the default cache mode and write-protect fields. The CACR is described in Section 5.2.1, "Cache Control Register (CACR)."

3.2.7 Access Control Registers (ACR*n*)

The access control registers define attributes for user-defined memory regions. These attributes include the definition of cache mode, write protect, and buffer write enables. The ACRs are described in Section 5.2.2, "Access Control Registers (ACR0–ACR1)."

3.2.8 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in 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 MByte boundary.

BDM: 0x801 (VBR)   Access: Supervisor read/write
                                                                                    BDM read/write
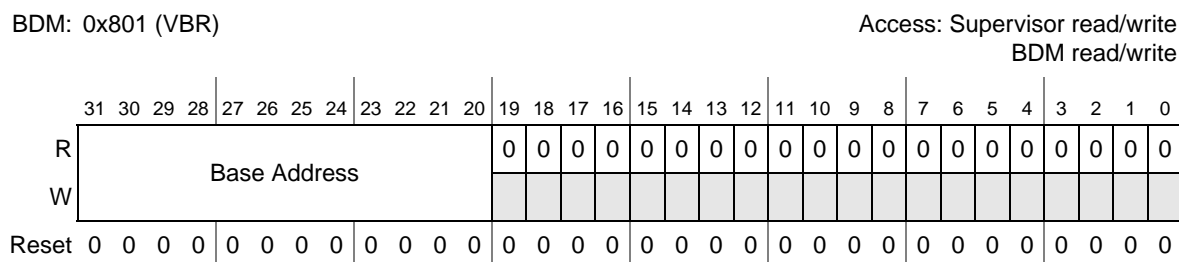
Figure 3-7. Vector Base Register (VBR)

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. 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
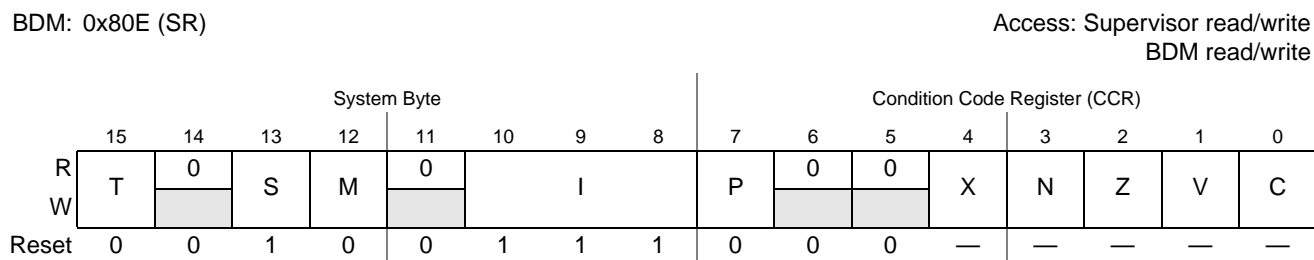
Figure 3-8. Status Register (SR)

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.

Table 3-3. SR Field Descriptions (continued)

Field	Description
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.
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 6.2.1, "SRAM Base Address Register (RAMBAR)".

3.3 Functional Description

3.3.1 Version 3 ColdFire Microarchitecture

The following diagrams present a more detailed view of the internal pipeline structures for the Version 3 design. In particular, note the increased length of the IFP with the early decode (ED) table lookup and the branch acceleration target address adders in the IED stage with the feedback to the prefetch address logic in the IAG stage. The OEP is essentially unchanged from the Version 2 design with the exception of the extended opword provided from the IFP as part of the instruction interface:

IAG	IC 1	IC 2	IED	IB
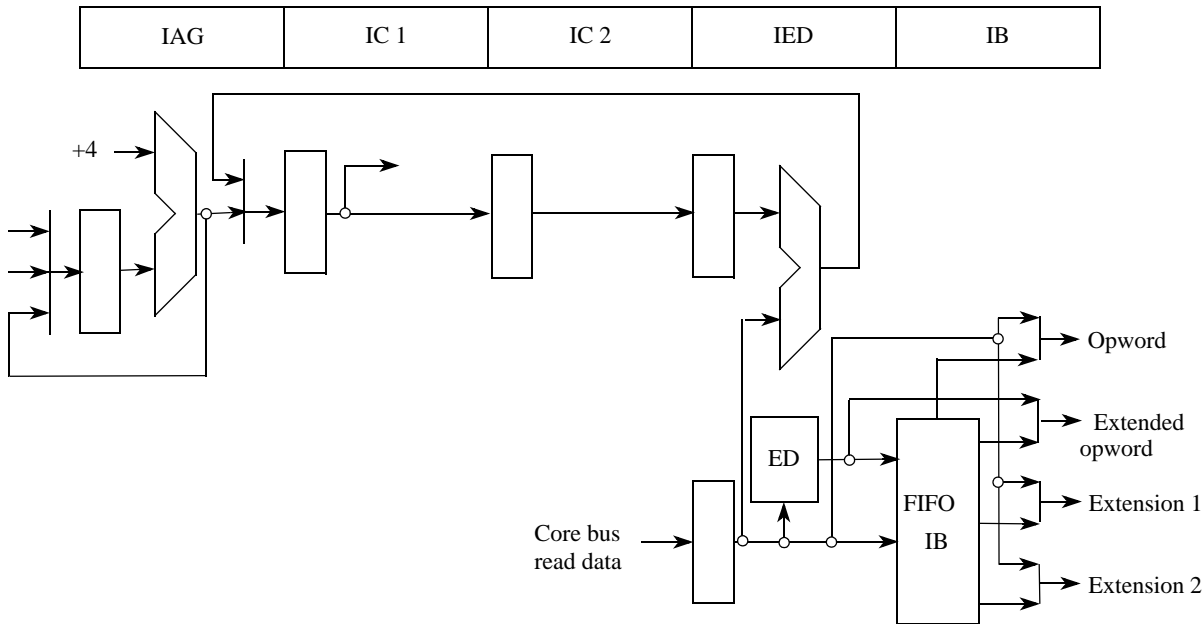
Figure 3-9. Version 3 ColdFire Processor Instruction Fetch Pipeline Diagram

DSOC	AGEX
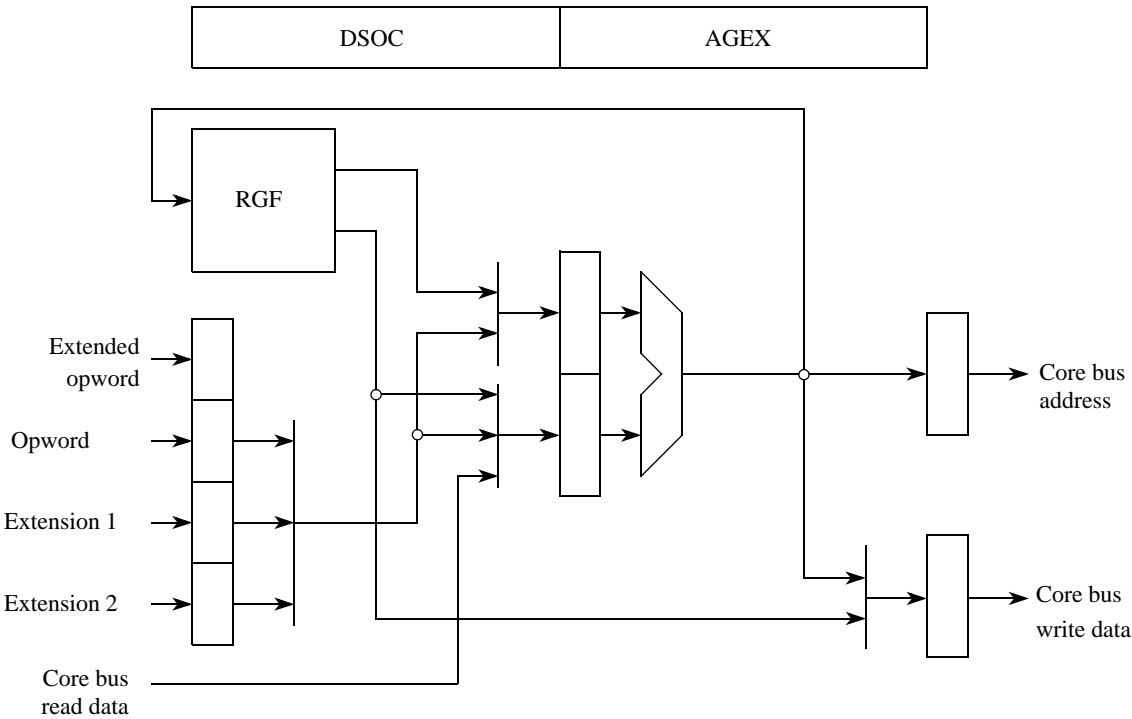
Figure 3-10. Version 3 ColdFire Processor Operand Execution Pipeline Diagram

3.3.2   Instruction Set Architecture (ISA_A+)

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 new 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_A+. 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; 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; 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.
Move from USP	USP → Destination register
Move to USP	Source register → USP

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
- Use of separate system stack pointers for user and supervisor modes.

All ColdFire processors use an instruction restart exception model. However, Version 3 ColdFire processors require more software support to recover from certain access errors. See Section 3.3.4.1, "Access Error Exception" for details.

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 3-11, the processor uses a simplified fixed-length stack frame for all exceptions. 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 Mbyte 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 ($4 \times$ 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 14, "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

Table 3-5. Exception Vector Assignments (continued)

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
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	Debug interrupt
13	0x034	—	Reserved
14	0x038	Fault	Format error
15–23	0x03C–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064–0x07C	—	Reserved
32–47	0x080–0x0BC	Next	Trap # 0-15 instructions
48–63	0x0C0–0x0FC	—	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 3-11 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			
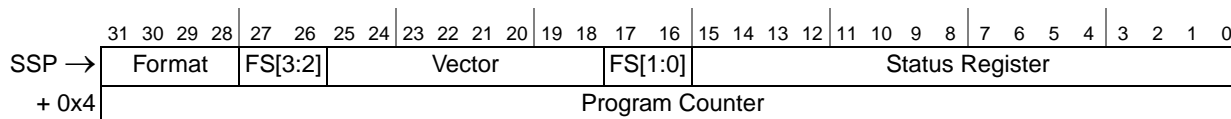
Figure 3-11. Exception Stack Frame Form

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
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

- 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 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. In this situation, any address register updates attributable to the auto-addressing modes, (for example, (An)+,-(An)), have already been performed, so the programming model contains the updated An value. In addition, if an access error occurs during a MOVEM instruction loading from memory, any registers already updated before the fault occurs contain the operands from memory.

The V3 ColdFire processor uses an imprecise reporting mechanism for access errors on operand writes. Because the actual write cycle may be decoupled from the processor's issuing of the operation, the signaling of an access error appears to be decoupled from the instruction that generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled. All programming model updates associated with the write instruction are completed. The NOP instruction can collect access errors for writes. This instruction delays its execution until all previous operations, including all pending write operations, are complete. If any previous write terminates with an access error, it is guaranteed to be reported on the NOP instruction.

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 3 ColdFire processor calculates the target address then the return address is pushed onto the stack. If an address error occurs on an RTS instruction, the Version 3 ColdFire processor overwrites the faulting return PC with the address error stack frame.

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 3-12. The opword line definition is shown in Table 3-8.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Line				OpMode						Effective Address					
										Mode			Register		
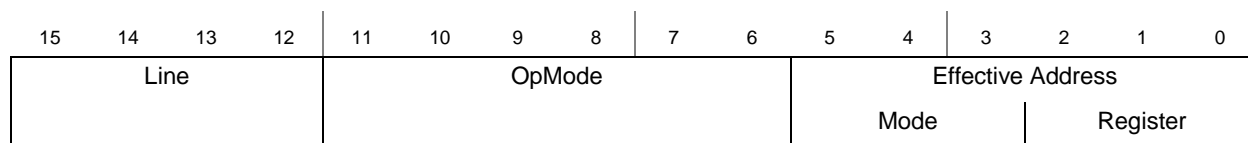
Figure 3-12. ColdFire Instruction Operation Word (Opword) Format

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)
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 Interrupt

See Chapter 36, "Debug Module," for a detailed explanation of this exception, which is generated in response to a hardware breakpoint register trigger. The processor does not generate an IACK cycle, but rather calculates the vector number internally (vector number 12). Additionally, SR[M,I] are unaffected by the interrupt.

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 14, "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 3-13.

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			
Reset	1	1	0	0	1	1	1	1	0	0	1	1	0	0	0	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			
Reset	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	1

Figure 3-13. 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 (This is the value used for this device.) 0100 V4 ColdFire core 0101 V5 ColdFire core Else Reserved for future use
19–16 REV	Processor revision number. The default is 0b0000.
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.)
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.
10–8	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 1000 ISA_A+ (This is the value used for this device.) 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+ (This is the value used for this device.) 1011 DEBUG_D+ 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)   Access: User read-only
      Store: 0x181 (D1)   BDM read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CLSZ		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	MBSZ		UCAS		UCSZ				SRAMSZ				0	0	0	
W																
Reset	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0
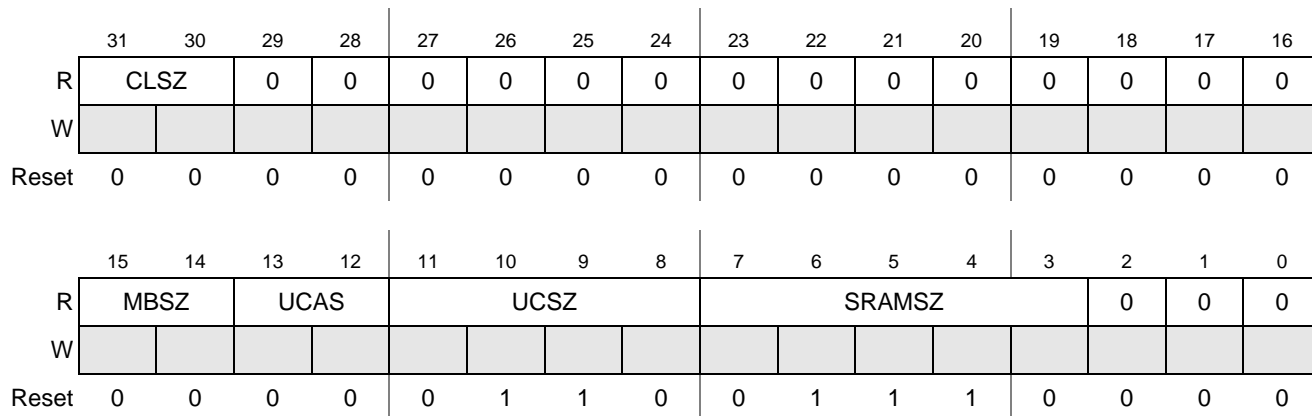
Figure 3-14. D1 Hardware Configuration Info

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.
29–24	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
13–12 UCAS	Unified cache associativity. Defines the unified cache set-associativity. 00 Four-way (This is the value used for this device) 01 Direct mapped Else Reserved for future use
11–8 UCSZ	Unified cache size. Indicates the size of the unified cache. 0000 No unified cache 0001 512 bytes 0010 1 Kbytes 0011 2 Kbytes 0100 4 Kbytes 0101 8 Kbytes 0110 16 Kbytes (This is the value used for this device) 0111 32 Kbytes 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 Kbytes 00110 2 Kbytes 01000 4 Kbytes 01010 8 Kbytes 01100 16 Kbytes 01110 32 Kbytes (This is the value used for this device) 10000 64 Kbytes 10010 128 Kbytes Else Reserved for future use
2–0	Reserved.

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. The OEP does not experience any sequence-related pipeline stalls. The most common example of stall involves consecutive store operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the processor are marked as busy for two clock cycles after the final decode and select/operand fetch cycle (DSOC) of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it is stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is two cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.

3. 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.

4. 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 3-11.

Table 3-11. 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 3-12 lists execution times for MOVE.{B,W} instructions; Table 3-13 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 3-12. 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)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1))	4(1/1)
(Ay)+	4(1/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1))	4(1/1)
-(Ay)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1))	4(1/1)
(d16,Ay)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—
(d8,Ay,Xi*SF)	5(1/0)	5(1/1)	5(1/1)	5(1/1)	—	—	—
xxx.w	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
xxx.l	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—

Table 3-12. MOVE Byte and Word Execution Times (continued)

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
(d16,PC)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—
(d8,PC,Xi*SF)	5(1/0)	5(1/1)	5(1/1)	5(1/1))	—	—	—
#xxx	1(0/0)	2(0/1)	2(0/1)	2(0/1)	—	—	—

Table 3-13. 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)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(Ay)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d16,Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,Ay,Xi*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
xxx.w	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.l	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(d16,PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,PC,Xi*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
#xxx	1(0/0)	(0/1)	(0/1)	(0/1)	—	—	—

3.3.5.3　Standard One Operand Instruction Execution Times

Table 3-14. 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)	—	—	—	—	—	—	—

Table 3-14. One Operand Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
FF1	Dx	1(0/0)	—	—	—	—	—	—	—
NEG.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEGX.L	Dx	1(0/0)	—	—	—	—	—	—	—
NOT.L	Dx	1(0/0)	—	—	—	—	—	—	—
SCC	Dx	1(0/0)	—	—	—	—	—	—	—
SWAP	Dx	1(0/0)	—	—	—	—	—	—	—
TST.B	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
TST.W	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
TST.L	<ea>	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

3.3.5.4 Standard Two Operand Instruction Execution Times

Table 3-15. 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)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
ADD.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ADDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ADDQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ADDX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
AND.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
AND.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(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)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCHG	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCLR	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BSET	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BTST	Dy,<ea>	2(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
BTST	#imm,<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	—	—	—

Table 3-15. 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
CMP.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
CMPI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
DIVS.W	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
DIVU.W	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
DIVS.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
DIVU.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
EOR.L	Dy,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(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)
MOVEQ.L	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
OR.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
OR.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
REMS.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
REMU.L	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
SUB.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
SUB.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
SUBI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
SUBQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
SUBX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

3.3.5.5 Miscellaneous Instruction Execution Times

Table 3-16. Miscellaneous Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
CPUSHL	(Ax)	—	11(0/1)	—	—	—	—	—	—
LINK.W	Ay,#imm	2(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)	—	—	—	—	—	—	—

Table 3-16. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
MOVE.W	<ea>,SR	7(0/0)	—	—	—	—	—	—	7(0/0) [2]
MOVEC	Ry,Rc	9(0/1)	—	—	—	—	—	—	—
MOVEM.L	<ea>,and list	—	1+n(n/0)	—	—	1+n(n/0)	—	—	—
MOVEM.L	and list,<ea>	—	1+n(0/n)	—	—	1+n(0/n)	—	—	—
NOP		3(0/0)	—	—	—	—	—	—	—
PEA	<ea>	—	2(0/1)	—	—	2(0/1) [4]	3(0/1) [5]	2(0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STOP	#imm	—	—	—	—	—	—	—	3(0/0) [3]
TRAP	#imm	—	—	—	—	—	—	—	15(1/2)
TPF		1(0/0)	—	—	—	—	—	—	—
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPF.L		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	2(1/0)	—	—	—	—	—	—	—
WDDATA	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	—
WDEBUG	<ea>	—	5(2/0)	—	—	5(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 3-17. 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	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)[1]	—	—	—
MAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.W	Ry, Rx, <ea>, Rw, Raccx	—	3(1/0)	3(1/0)	3(1/0)	3(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	5(0/0)	—	—	—	—	—	—	5(0/0)

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

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MOVE.L	<ea>y, Rmask	4(0/0)	—	—	—	—	—	—	4(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]	—	—	—	—	—	—	—
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	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)[1]	—	—	—
MSAC.W	Ry, Rx, <ea>, Rw, Raccx	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)[1]	—	—	—
MULS.L	<ea>y, Dx	4(0/0)	7(1/0)	7(1/0)	7(1/0)	7(1/0)	—	—	—
MULS.W	<ea>y, Dx	4(0/0)	7(1/0)	7(1/0)	7(1/0)	7(1/0)	8(1/0)	7(1/0)	4(0/0)
MULU.L	<ea>y, Dx	4(0/0)	7(1/0)	7(1/0)	7(1/0)	7(1/0)	—	—	—
MULU.W	<ea>y, Dx	4(0/0)	7(1/0)	7(1/0)	7(1/0)	7(1/0)	8(1/0)	7(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 3-18. 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]	—	—	—
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		—	—	14(2/0)	—	—	—	—	—
RTS		—	—	8(1/0)	—	—	—	—	—

Table 3-19. Bcc Instruction Execution Times, CCR[P]=0

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
Bcc	5(0/0)	1(0/0)	1(0/0)[3]	5(0/0)

Table 3-20. Bcc Instruction Execution Times, CCR[P]=1

Opcode	Predicted Correctly as Taken	Predicted Correctly as Not Taken	Predicted Incorrectly
Bcc	1(0/0)	1(0/0)	5(0/0)[3]

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 conditional branch opcodes (bcc), a static algorithm is used to determine the prediction state of the branch. This algorithm is:

```
if bcc is a forward branch
        if CCR[P] == 0
```

```
                            bcc is predicted as not-taken
                else
                            bcc is predicted as taken
        else
                bcc is a backward branch and predicted as taken
```

The execution times in the BRA, Bcc (Table 3-19) assume that CCR[P] is cleared. Another representation of the Bcc execution times is shown in Table 3-20.

The execution time for the predicted correctly as taken column can vary between 1 to 3 cycles depending on the amount of decoupling between the IFP and OEP as previously discussed.

Chapter 4
Enhanced Multiply-Accumulate Unit (EMAC)

4.1 Introduction

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

4.1.1 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 4-1).
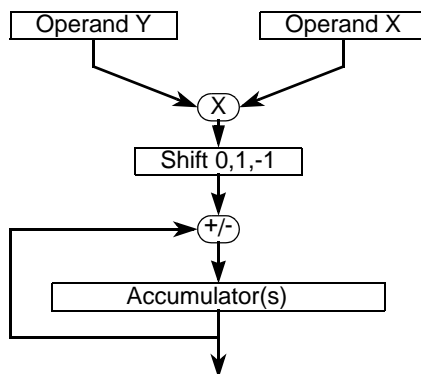
Figure 4-1. Multiply-Accumulate Functionality Diagram

4.1.1.1   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 4-1.

$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k) \qquad \textit{Eqn. 4-1}$$

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 4-1 to a simple, four-tap FIR filter, shown in Equation 4-2, in which the accumulated sum is a past data values and coefficients sum.

$$y(i) = \sum_{k=0}^{3} b(k)x(i-k) = b(0)x(i) + b(1)x(i-1) + b(2)x(i-2) + b(3)x(i-3) \qquad \textit{Eqn. 4-2}$$

4.2　Memory Map/Register Definition

The following table and sections explain the MAC registers:

Table 4-1. EMAC Memory Map

BDM[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	4.2.1/4-3
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	4.2.2/4-5
0x806	MAC Accumulator 0 (ACC0)	32	R/W	Undefined	4.2.3/4-6
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	4.2.4/4-7
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	4.2.4/4-7
0x809	MAC Accumulator 1 (ACC1)	32	R/W	Undefined	4.2.3/4-6
0x80A	MAC Accumulator 2 (ACC2)	32	R/W	Undefined	4.2.3/4-6
0x80B	MAC Accumulator 3 (ACC3)	32	R/W	Undefined	4.2.3/4-6

[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 36, "Debug Module."

4.2.1　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
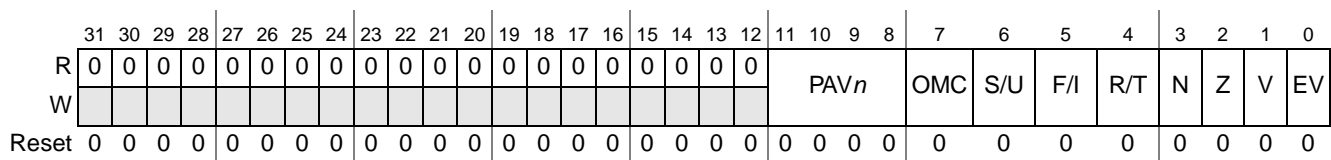
Figure 4-2. MAC Status Register (MACSR)

Table 4-2. 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 is used to form 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.

Table 4-2. MACSR Field Descriptions (continued)

Field	Description
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 4.3.1.1, "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 4.3.4, "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 4.3.1.1, "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.
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.

Table 4-2. MACSR Field Descriptions (continued)

Field	Description
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 4-3 summarizes the interaction of the MACSR[S/U,F/I,R/T] control bits.

Table 4-3. 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

4.2.2   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>yand ,Rw
```

The and 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 and  {0xFFFF, MASK}

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

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

        if <ea> = (d16,An)
                oa  = (An + se_d16) and  {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
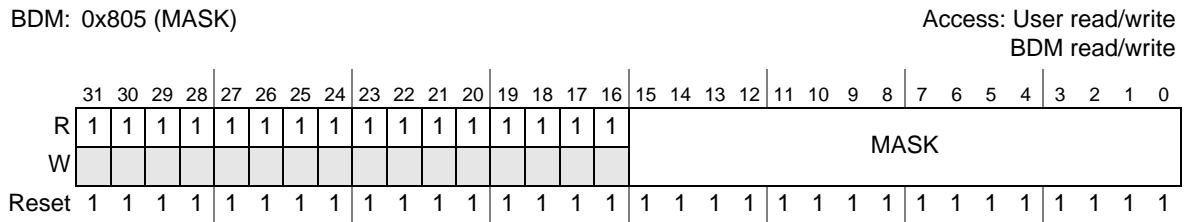
Figure 4-3. Mask Register (MASK)

Table 4-4. 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.

4.2.3   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)   Access: User read/write
      0x809 (ACC1)   BDM read/write
      0x80A (ACC2)
      0x80B (ACC3)

31 30 29 28	27 26 25 24	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 – – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
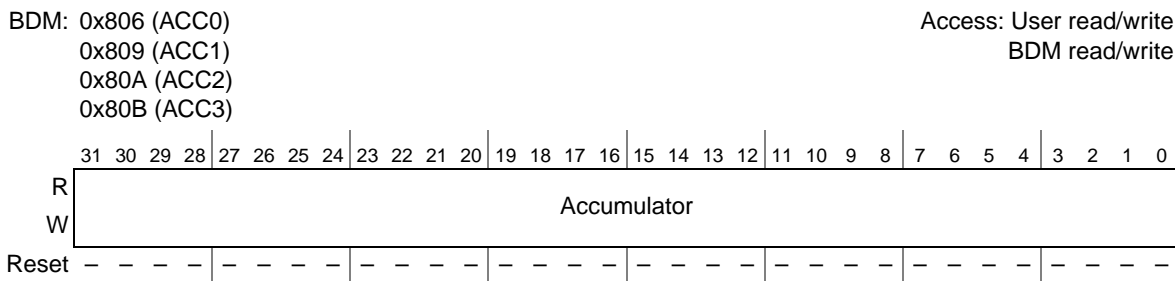
Figure 4-4. Accumulator Registers (ACC0–3)

Table 4-5. ACC0–3 Field Descriptions

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

4.2.4 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 4.3, "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 – – – – – – – –	– – – – – – – –	– – – – – – – –	– – – – – – – –
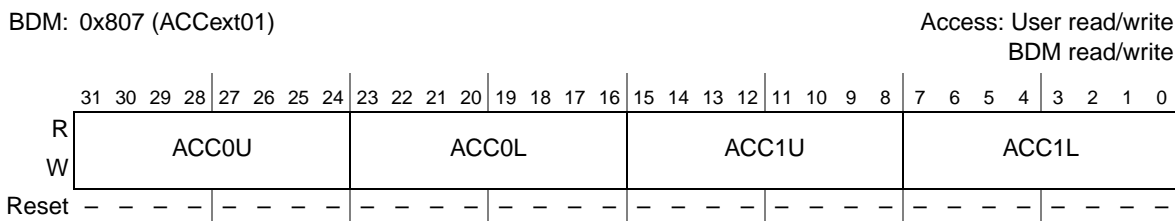
Figure 4-5. Accumulator Extension Register (ACCext01)

Table 4-6. 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	ACC2U		ACC2L		ACC3U		ACC3L	
W								
Reset	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
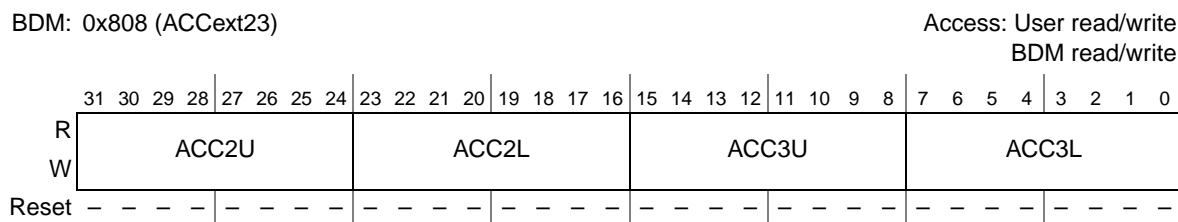
Figure 4-6. Accumulator Extension Register (ACCext23)

Table 4-7. 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

4.3 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.

and show relative alignment of input operands, the full 64-bit product, the resulting 40-bit product used for accumulation, and 48-bit accumulator formats.
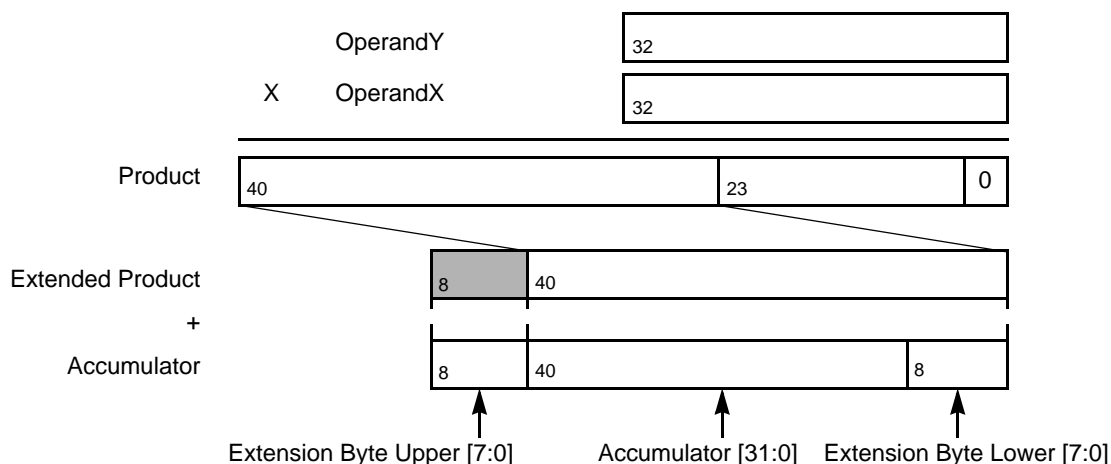
Figure 4-7. Fractional Alignment
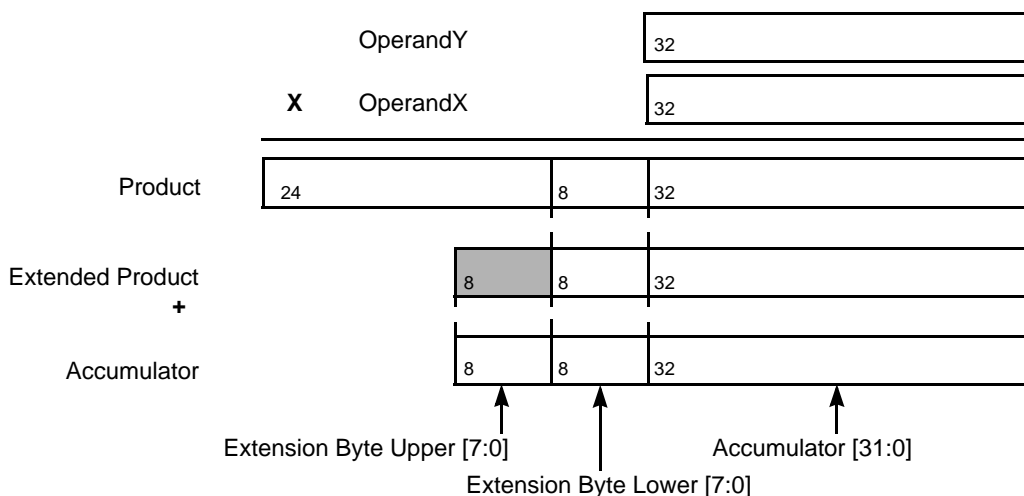
Figure 4-8. Signed and Unsigned Integer Alignment

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.

4.3.1 Fractional Operation Mode

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

4.3.1.1 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.

4.3.1.2 Saving and Restoring the EMAC Programming Model

The presence of rounding logic in the EMAC output datapath requires that 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.

4.3.1.3   MULS/MULU

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

4.3.1.4   Scale Factor in MAC or MSAC Instructions

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

4.3.2   EMAC Instruction Set Summary

Table 4-8 summarizes EMAC unit instructions.

Table 4-8. 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 4-8. 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

4.3.3 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 AGEX 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 4-9 shows EMAC timing.
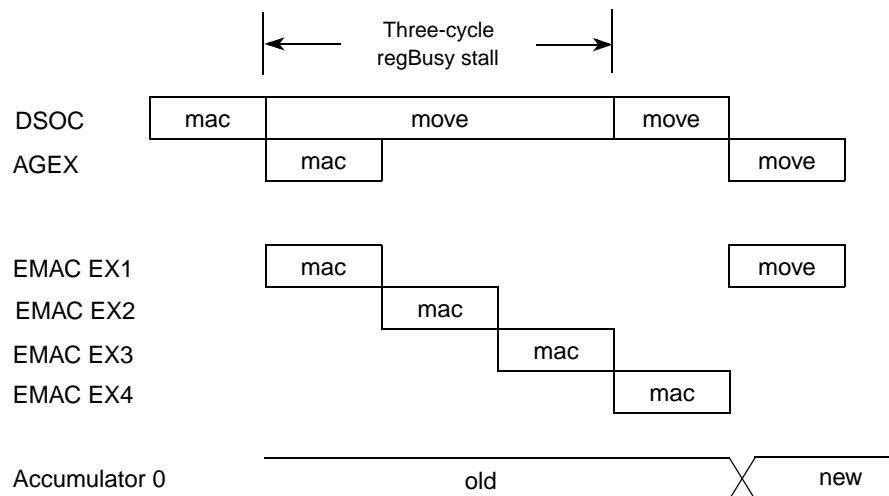
Figure 4-9. EMAC-Specific OEP Sequence Stall

In Figure 4-9, the OEP stalls the store-accumulator instruction for three cycles: the EMAC pipleine depth minus 1. The minus 1 factor is needed because the OEP and EMAC pipelines overlap by a cycle, the AGEX stage. As the store-accumulator instruction reaches the AGEX 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.

4.3.4   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 $-2^{(N-1)}$ $\leq$ 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 $0 \leq$ 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, $a_{N-1}a_{N-2}a_{N-3}... a_2a_1a_0$, its value is given by the equation in Equation 4-3.

$$value = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{-(i+1-N)} \cdot ai$$

Eqn. 4-3

This format can represent numbers in the range $-1 \leq$ 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 $(1 - 2^{-31})$.

4.3.5   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 4.2.1, "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) and and  (product[63:39] != 0xffff_ff_1))
              then {          /* product overflow */
                    MACSR.PAVn = 1
                    MACSR.V = 1
                    if (inst == MSAC and and  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) and and  (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) and and  (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 and and  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 and and  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 5
Cache

5.1   Introduction

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

5.1.1   Overview

This ColdFire processor contains a non-blocking, 16-Kbyte, 4-way set-associative, unified (instruction and data) cache with a 16-byte line size. The cache improves system performance by providing low-latency 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. Figure 5-1 shows the organization and integration of the cache.
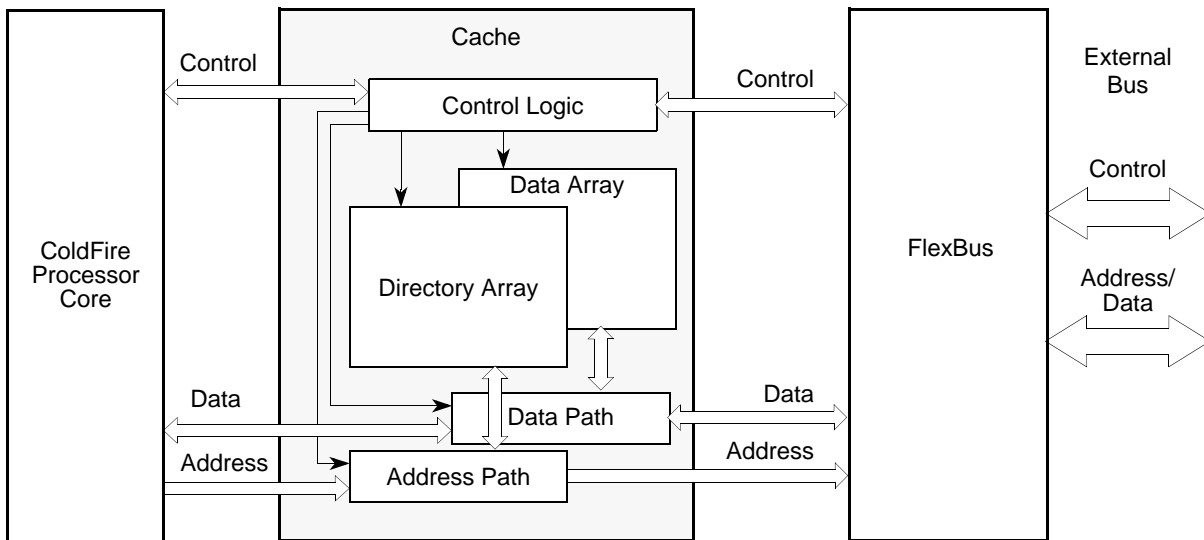
Figure 5-1. Unified Cache Organization

The cache supports operation of copyback, write-through, or cache-inhibited modes. A nonblocking cache services read hits or write hits from the processor while a fill (caused by a cache allocation) is in progress. As Figure 5-1 shows, instruction and data 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. In write, 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 device does not implement bus snooping; cache coherency with other possible bus masters must be maintained in software.

5.2 Memory Map/Register Definition

This section describes the implementation of the cache registers.

Table 5-1. Cache Programming Model

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	5.2.1/5-2
0x004	Access Control Register 0 (ACR0)	32	R/W	Undefined	Yes	5.2.2/5-4
0x005	Access Control Register 1 (ACR1)	32	R/W	Undefined	Yes	5.2.2/5-4

[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 36, "Debug Module."

5.2.1 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 the tags, state information, or data in the cache.

BDM: 0x002 (CACR)
Access: MOVEC write-only
Debug read/write
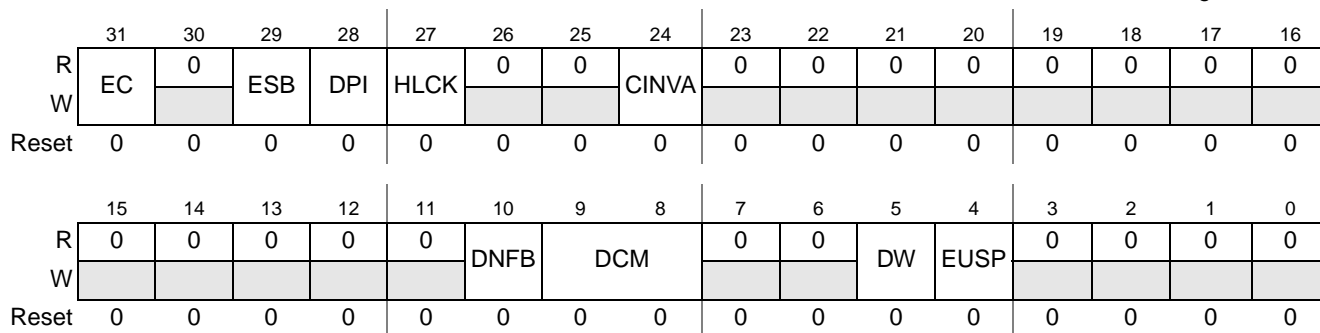
Figure 5-2. Cache Control Register (CACR)

Table 5-2. CACR Field Descriptions

Field	Description
31 EC	Enable cache. 0 Cache disabled. The data cache is not operationa;l however, data and tags are preserved. 1 Cache enabled.
30	Reserved, should be cleared.

Table 5-2. CACR Field Descriptions (continued)

Field	Description
29 ESB	Enable store buffer. 0 All writes to write-through or noncachable imprecise space bypass the store buffer and generate bus cycles directly. The associated performance penalty is described in Section 5.3.7.2.1, "Push and Store Buffers." 1 The four-entry FIFO store buffer is enabled; this buffer defers pending writes to write-through or cache-inhibited imprecise regions to maximize performance. Accesses to cache-inhibited, precise memory always bypass the store buffer.
28 DPI	Disable CPUSHL invalidation. 0 Normal operation. A CPUSHL instruction causes the selected line to be pushed if modified and then invalidated. 1 No clear operation. A CPUSHL instruction causes the selected line to be pushed if modified, then left valid.
27 HLCK	Half cache lock mode 0 Normal operation. The cache allocates the lowest non-valid way. If all ways are valid, the cache allocates the way pointed at by the counter and then increments this counter modulo-4. 1 Half cache operation. The cache allocates to the lowest non-valid way of levels 2 and 3; if both 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 modulo-2. This locks the content of ways 0 and 1. Ways 0 and 1 remain updated on write hits and may be pushed and/or cleared by specific cache push/invalidate instructions. This implementation allows maximum use of the available cache memory and also provides the flexibility of setting HLCK before, during, or after the needed allocations occur.
26–25	Reserved, should be cleared.
24 CINVA	Cache invalidate all. Writing a 1 to this bit initiates entire cache invalidation. After invalidation is complete, this bit automatically returns to 0; it is not necessary to clear it explicitly. This bit is always read as a 0. Caches are not cleared on power-up or normal reset, as shown in Figure 5-5. 0 No invalidation is performed 1 Initiate invalidation of the entire 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.
23–11	Reserved, should be cleared.
10 DNFB	Default noncacheable fill buffer 0 Fill buffer not used to store noncacheable instruction accesses (16 or 32 bits). 1 Fill buffer used to store noncacheable accesses. The fill buffer is used only for normal (TT = 0) instruction reads of a noncacheable region. Instructions are loaded into the fill buffer by a burst access (same as a line fill). They stay in the buffer until they are displaced, so subsequent accesses may not appear on the external bus. **Note:** This feature can cause a coherency problem for self-modifying code. If enabled and a cache-inhibited access uses the fill buffer, instructions remain valid in the fill 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 of that written data are serviced by the fill buffer and receive stale information.
9–8 DCM	Default cache mode. Selects the default cache mode and access precision as follows: 00 Cacheable, write-through 01 Cacheable, copy-back 10 Cache-inhibited, precise exception model 11 Cache-inhibited, imprecise exception model. Precise and imprecise accesses are described in Section 5.3.4, "Cache-Inhibited Accesses."
7–6	Reserved, should be cleared.
5 DW	Default write protect. Indicates the default write privilege. 0 Read and write accesses permitted 1 Write accesses not permitted

Table 5-2. CACR Field Descriptions (continued)

Field	Description
4 EUSP	Enable user stack pointer. See Section 3.2.3, "Supervisor/User Stack Pointers (A7 and OTHER_A7)," for more information on the dual stack pointer implementation. 0 Disable the processor's use of the User Stack Pointer 1 Enable the processor's use of the User Stack Pointer
3–0	Reserved, should be cleared.

5.2.2   Access Control Registers (ACR0–ACR1)

The access control registers (ACRn) assign control attributes, such as cache mode and write protection, to specified memory regions. Registers are accessed with the MOVEC instruction with the encoding shown in Figure 5-3.

For overlapping regions, ACR0 takes priority. Data transfers to and from these registers are longword transfers. Bits 12–7, 4, 3, 1, and 0 are always read as zeros.

NOTE

Peripheral space (0xE000_0000–0xFFFF_FFFF) should not be cached. The combination of the CACR defaults and the two ACRn registers must define the non-cacheable attribute for this address space.

BDM: 0x004 (ACR0)   Access: MOVEC write-only
0x005 (ACR1)   Debug read/write
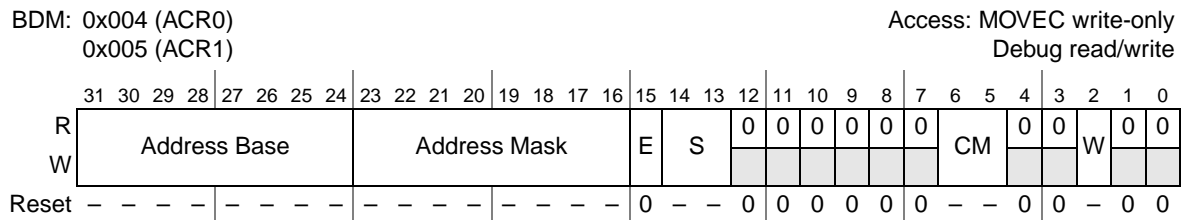
Figure 5-3. Access Control Register Format (ACRn)

Table 5-3. ACRn Field Descriptions

Field	Description
31–24 Address Base	Address base. Compared with address bits A[31:24]. Eligible addresses that match are assigned the access control attributes of this register.
23–16 Address Mask	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 noncontiguous regions of memory.
15 E	Enable. Enables or disables the other ACRn bits. 0 Access control attributes disabled 1 Access control attributes enabled

Table 5-3. ACR*n* Field Descriptions (continued)

Field	Description
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
12–7	Reserved, should be cleared.
6–5 CM	Cache mode. Selects the cache mode and access precision. Precise and imprecise accesses are described in Section 5.3.4, "Cache-Inhibited Accesses." 00 Cacheable, write-through 01 Cacheable, copyback 10 Cache-inhibited, precise 11 Cache-inhibited, imprecise
4–3	Reserved, should be cleared.
2 W	Write protect. Selects the write privilege of the memoryregion. 0 Read and write accesses permitted 1 Write accesses not permitted
1–0	Reserved, should be cleared.

5.3   Functional Description

5.3.1   Cache Organization

A four-way set associative cache is organized as four ways (levels). There are 256 sets in the 16-Kbyte 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). 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 5-4 shows cache organization and terminology used.
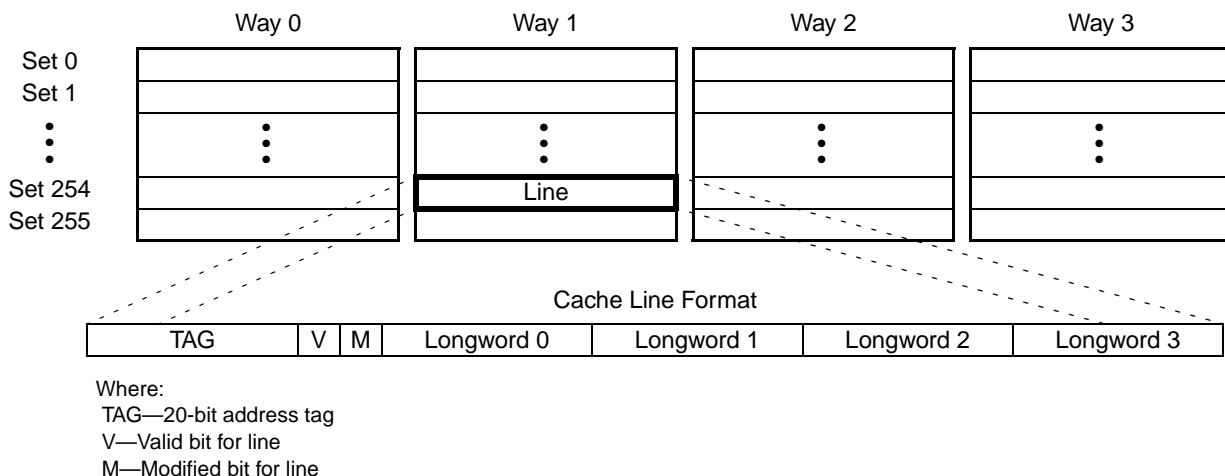
Figure 5-4. Data Cache Organization and Line Format

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

As shown in Table 5-4, a cache line is always in one of three states: invalid, valid-unmodified (often referred to as exclusive), or valid-modified.

Table 5-4. Valid and Modified Bit Settings

V	M	Description
0	x	Invalid. Invalid lines are ignored during lookups.
1	0	Valid, not modified. 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.

A valid line can be explicitly invalidated by executing a CPUSHL instruction**.**

5.3.1.2 Cache at Start-Up

As Figure 5-5 (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[CINVA] 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 5-5 (D). This process is described in detail in Section 5.3, "Functional Description."

Figure 5-5. Data Cache: A) at Reset; B) after Invalidation; C and D) Loading Pattern

5.3.2 Cache Operation

Figure 5-6 shows the general flow of a caching operation.
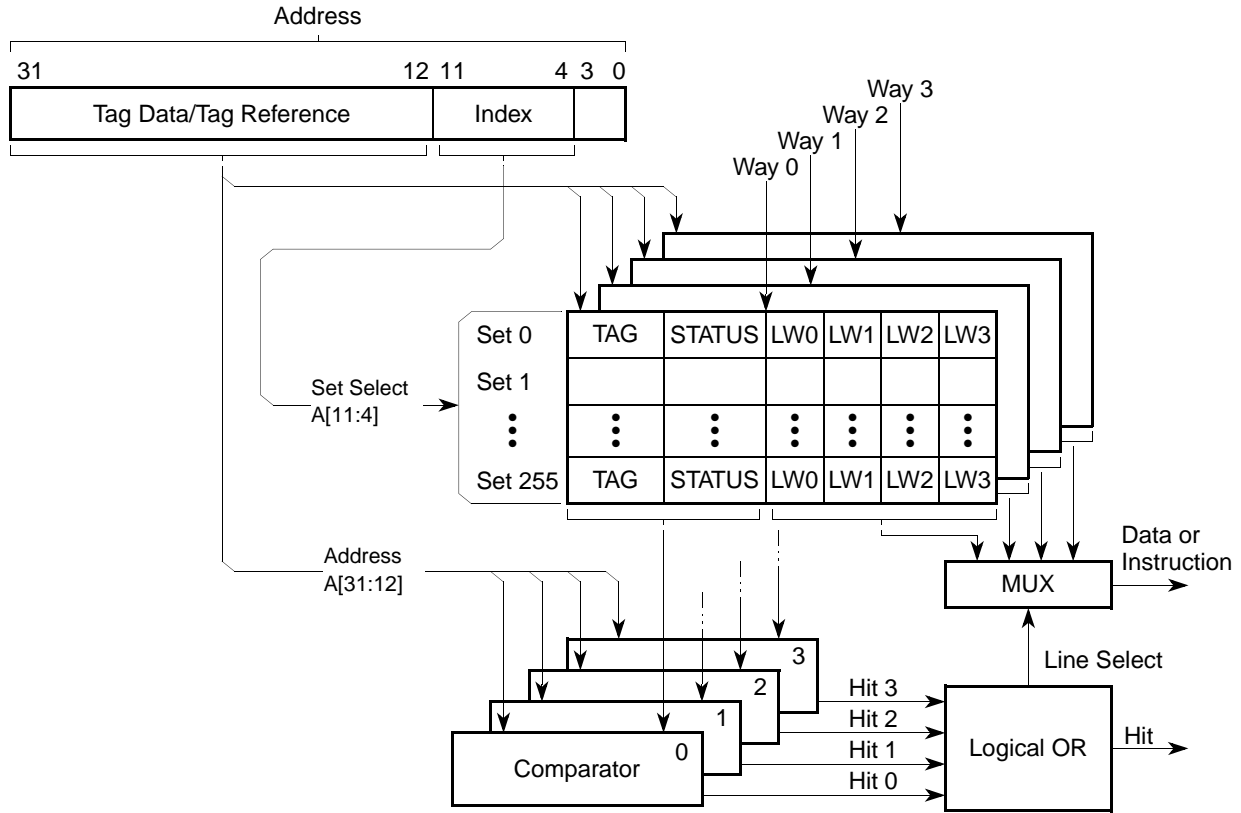
Figure 5-6. Data Caching Operation

The following steps determine if a 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 are used to update the cache line tag field. A[31:12] can specify 20 possible addresses 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 contain valid data. If a cacheable write access hits in a valid cache line, the write can occur to the cache line without having to load it from memory.

    If the memory space is copyback, the updated cache line is marked modified (M = 1), because the new data has made the data in memory out of date. If the memory location is write-through, the write is passed on to system memory and the M bit is never 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 is used to choose 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[HLCK] = 1, 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, the following 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.

The following exceptions apply:

- 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 = 1 and M = 1).
- 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 to this normal operation 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 (that is, transfer type (TT) equals 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 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, it is generally recommended to use the CPUSHL instruction to push and/or invalidate the cache entry or set CACR[CINVA] to invalidate the cache before switching cache modes.

5.3.3   Caching Modes

Caching modes determine how the cache manages an access. An access can be cacheable in write-through or copyback mode; it can be cache-inhibited in precise or imprecise modes. For normal accesses, the ACRn[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[DCM].

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 5-5, reset does not automatically invalidate cache entries; they must be invalidated through software.

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.

5.3.3.1   Cacheable Accesses

If ACRn[CM] or the default field of the CACR indicates write-through or copyback, the access is cacheable. A read access to a write-through or copyback region is read from the cache if matching data is found. Otherwise, the data is read from memory and the cache is updated. When a line is being read from memory for 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.

5.3.3.2   Write-Through Mode

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[ESB]. Writes in write-through mode are managed 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 be loaded 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 be loaded into the cache.

5.3.3.3   Copyback Mode

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.

Be sure to flush the cache using the CPUSHL instruction before invalidating the cache in copyback mode. 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.

5.3.4   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 memory mapped registers. If the corresponding ACRn[CM] or CACR[DCM] 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.

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 (that is, 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 ACRn[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[CINVA] to invalidate the entire cache.

If ACRn[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 in the cache to occur before completion of a pending write from a previous instruction. Writes are not deferred past data-read accesses that miss the cache (that is, that 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 modem an exception aborts the instruction and the data may be accessed again when the instruction is restarted. These guarantees apply only when ACRn[CM] indicates precise mode and aligned accesses.

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

5.3.5   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).

5.3.5.1 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.

5.3.5.2 Write Miss

The cache controller manages processor writes that miss in the 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 5-7.

1. Write character X to 0x0B generates a write miss. Write cannot take place to an invalid line.
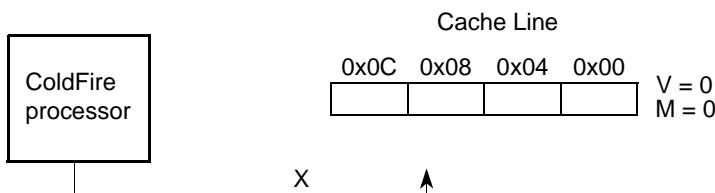
2. The cache line (characters A–P) is updated from system memory and line is marked Valid.
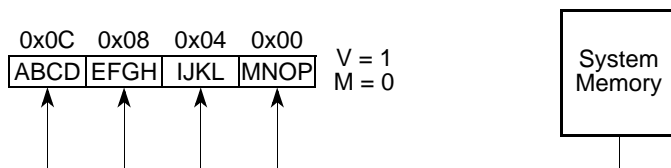
3. After the cache line is filled, the write that initiated the write -miss (the character X) completes to 0x0B.
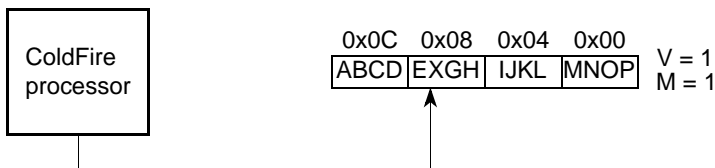
Figure 5-7. Write-Miss in Copyback Mode

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.

5.3.5.3 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[CINVA] before switching the cache mode.

5.3.5.4 Write Hit

The cache controller manages processor writes that hit in the 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.

5.3.6 Cache Coherency

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 (that is, cache coherency is not supported while external or DMA masters are using the bus). Therefore, on-chip DMA channels should not access cached local memory locations, as coherency is not maintained with the unified cache.

5.3.7 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.

5.3.7.1 Cache Filling

When a new cache line is required, a line read is requested, which generates a burst-read transfer. The responding device supplies 4 longwords of data in sequence. Burst operations can be inhibited or enabled through the burst read enable and burst write enable bits (BSTR and BSTW) in the chip-select control registers (CSCR0–CSCR7).

Line accesses implicitly request burst-mode operations from memory. For more information regarding external bus burst-mode accesses, see Chapter 17, "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 is aborted by an 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."

5.3.7.2 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 is written back to memory and the push buffer is invalidated.

5.3.7.2.1 Push and Store Buffers

The 16-byte push buffer reduces latency for requested new data on a cache miss by holding a displaced modified 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.

The store buffer implements a FIFO buffer that can defer pending writes to imprecise regions 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 of retire those writes. Writes to imprecise memory 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 (that is, store buffer disabled or cache-inhibited precise mode), external bus cycles are generated 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 5 cycles, making the minimum write time equal to 6 cycles when the store buffer is not used. See Section 3.1.1, "Overview."

The store buffer enable bit, CACR[ESB], controls the enabling of the store buffer. This bit can be set and cleared by the MOVEC instruction. At reset, this bit is cleared and all writes are precise. ACR[CM] or CACR[DCM] generates the mode used when this bit is set. The cacheable write-through and the cache-inhibited imprecise modes use the store buffer.

The store buffer can queue data to as much as four bytes wide per entry. Each entry matches a 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 to an odd-byte boundary—one per bus cycle.

5.3.7.2.2 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 that 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 are 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.

5.3.8 Cache Locking

Ways 0 and 1 of the cache can be locked by setting CACR[HLCK]. If the cache is locked, cache lines in ways 0 and 1 are not subject to deallocation by normal cache operations.

As Figure 5-8 (B and C) shows, the algorithm for updating the cache and for identifying cache lines to be deallocated is otherwise unchanged. 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 remain updated on write hits (D in Figure 5-8) 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[HLCK] 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

After reset, the cache is invalidated, then ways 0 and 1 are written with data that should not be deallocated.

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

While the cache is locked, after a position in way 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 5-8. Cache Locking

5.3.9 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[CINVA] to invalidate the cache before enabling it.

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[DPI] 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). The state of CACR[EC] does not affect the operation of CPUSHL or CACR[CINVA]. Disabling a cache by setting 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 where a physical address is specified. Figure 5-9 shows the A*x* format.

31	30	29	28	27	26	25	24	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	0			Set Index							Line Index		

Figure 5-9. A*x* Format

Bits A[11:4] specify a set index and bits A[3:0] specify the cache way.

5.3.10   Cache Operation Summary

This section gives operational details for the cache and presents cache-line state diagrams. Using the V and M bits, the cache supports a line-based protocol allowing individual cache lines to be in one of three states: invalid, valid, or modified. To maintain coherency with memory, the cache supports write-through and copyback modes, specified by the corresponding ACR[CM], or CACR[DCM] 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 temporarily buffered and later copied back to memory after the new line has been read from memory.

Figure 5-10 shows the three possible 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 5-5.
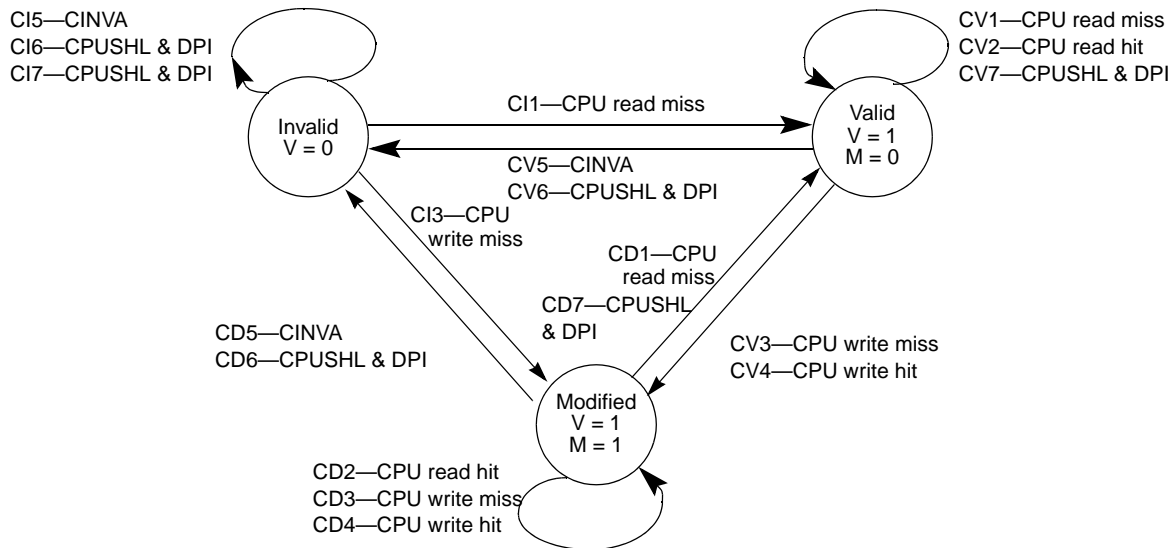
Figure 5-10. Cache Line State Diagram—Copyback Mode

Figure 5-11 shows the two possible states for a cache line in write-through mode.
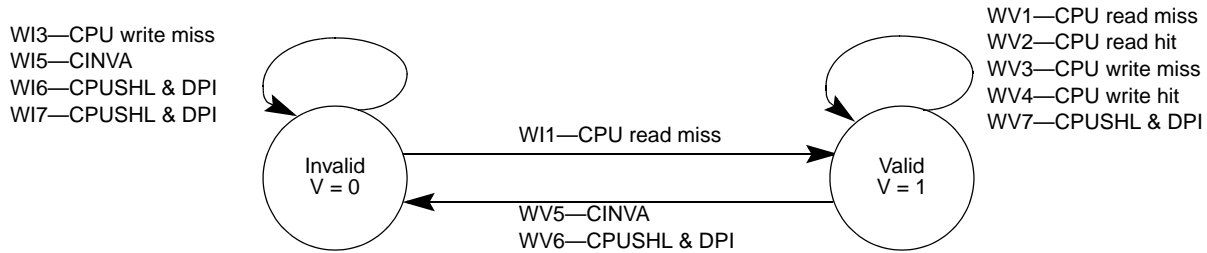
Figure 5-11. Cache Line State Diagram—Write-Through Mode

Table 5-5 describes cache line transitions and the accesses that cause them.

Table 5-5. 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.

Table 5-5. Cache Line State Transitions (continued)

Access	Current State					
	Invalid (V = 0)		Valid (V = 1, M = 0)		Modified (V = 1, M = 1)	
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[CINVA] 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[CINVA] 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	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.
Cache push	(C,W)I7	No action; stay in 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 5-5, organized by the current state of the cache line. In Table 5-6 the current state is invalid.

Table 5-6. 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 (copy-back)	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 (copy-back)	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 5-7 the current state is valid.

Table 5-7. 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 (copy-back)	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 (copy-back)	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 5-7. 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 5-8 the current state is modified.

Table 5-8. 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 (copy-back)	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[CINVA] before switching modes.
Write hit (copy-back)	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[CINVA] 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

Chapter 6
Static RAM (SRAM)

6.1 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.

6.1.1 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 11, "System Control Module (SCM)."

6.1.2 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

6.2   Memory Map/Register Description

The SRAM programming model shown in Table 6-1 includes a description of the SRAM base address register (RAMBAR), SRAM initialization, and power management.

Table 6-1. 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	6.2.1/6-2

[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 36, "Debug Module."

6.2.1   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 adress 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 6-1.

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	0	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
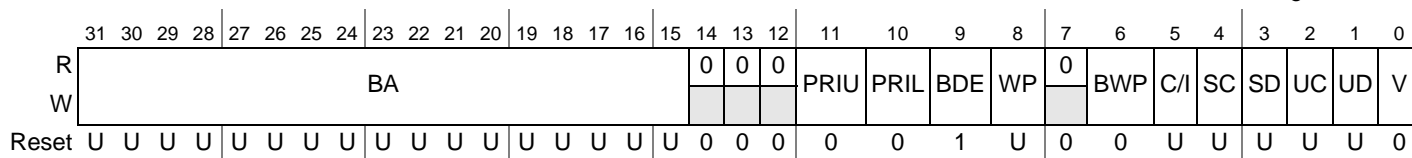
Figure 6-1. SRAM Base Address Register (RAMBAR)

Table 6-2. 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: {{TABLE}} **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 reaought d 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	Reserved, must be cleared.
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.

Embedded table within field 11–10:

PRIU,PRIL	Upper Bank Priority	Lower Bank Priority
00	SRAM Backdoor	SRAM Backdoor
01	SRAM Backdoor	CPU
10	CPU	SRAM Backdoor
11	CPU	CPU

Table 6-2. RAMBAR Field Descriptions (continued)

Field	Description
5–1 C/I, SC, SD, UC, UD	Address Space Masks (ASn). 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 6.3.2, "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

6.3   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.

6.3.1   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
```

6.3.2   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 ASn bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. Table 6-3 shows examples of typical RAMBAR settings.

Table 6-3. Typical RAMBAR Setting Examples

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

Chapter 7
Clock Module

7.1 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 be used to clock the device directly. The clock module contains:

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

NOTE

Throughout this manual, f_{sys} refers to the core frequency and $f_{sys/3}$ refers to the internal bus frequency.

Figure 7-1 is a high level representation of the clock connections. The exact functionality of the blocks is not illustrated (e.g. clocks to the SDRAMC and USB are disabled when the device is in limp mode, and the clocks to individual modules may be disabled via the peripheral power management registers).
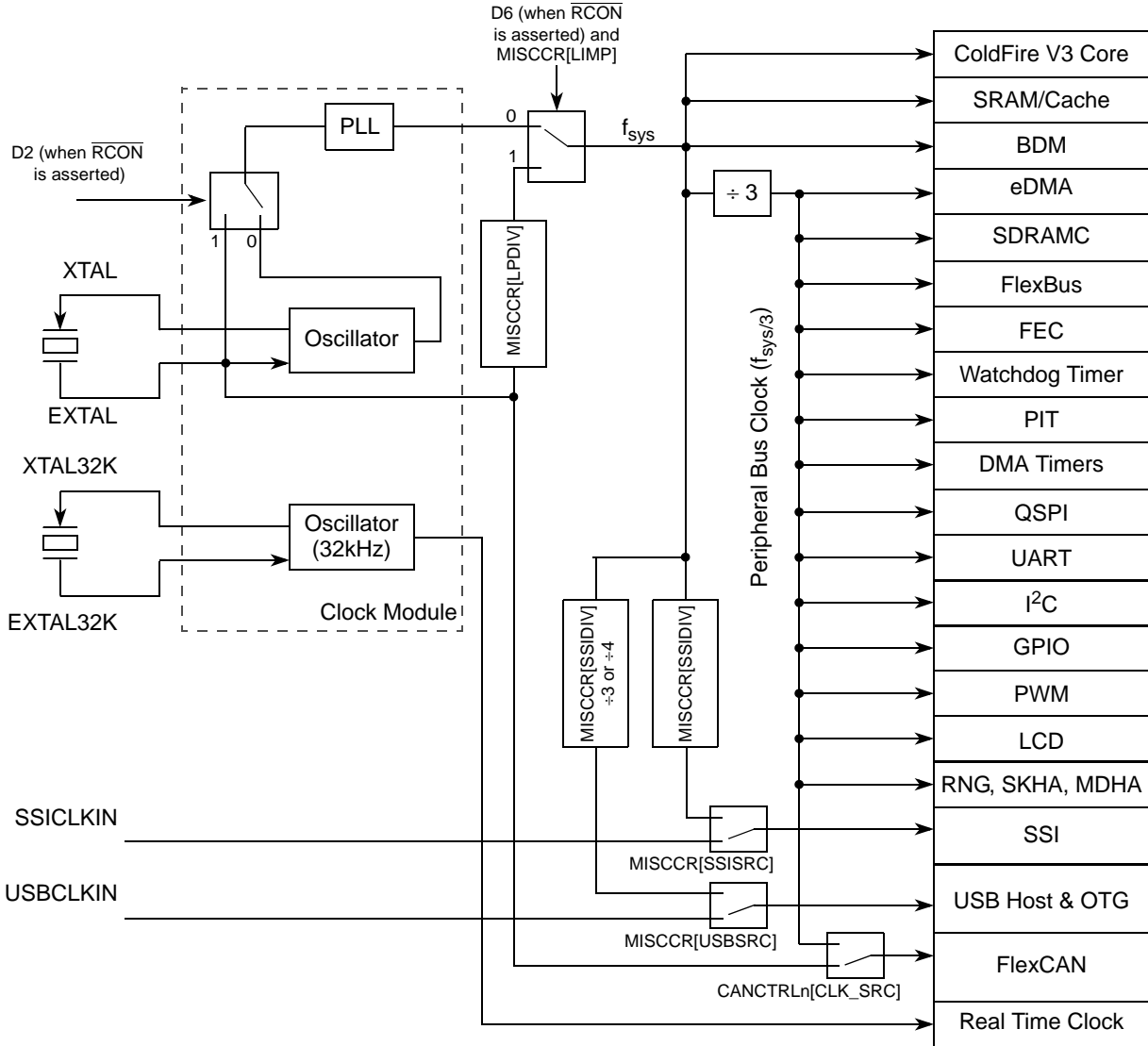
Figure 7-1. Device Clock Connections

7.1.1 Block Diagram

Figure 7-2 shows a block diagram of the clock module.
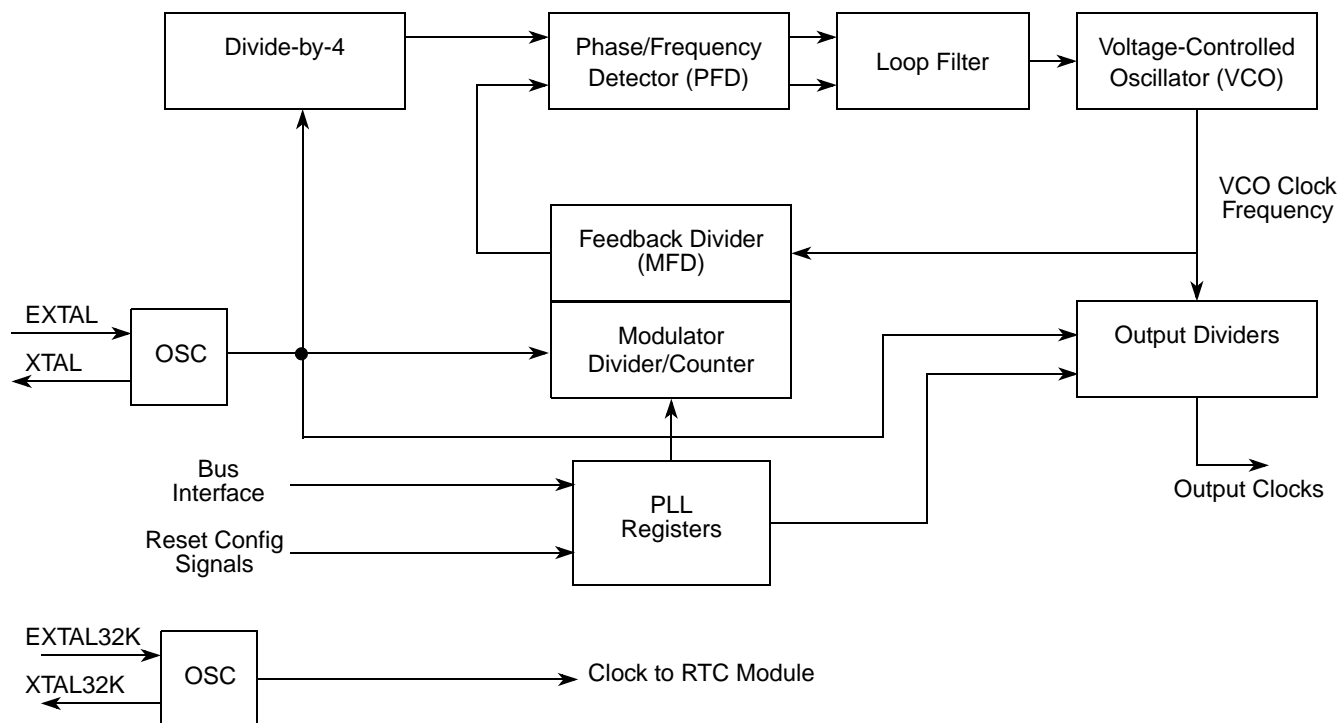
Figure 7-2. Clock Module Diagram

7.1.2 Features

Features of the clock module include the following:

- 16-MHz reference crystal oscillator
- Voltage controlled oscillator range from 350 MHz to 540 MHz, resulting in a core frequency ($f_{vco} \div 2$) of 175 MHz to 240 MHz (maximum rated for device)
- Programmable dithering
- Support for low-power modes
- Direct clocking of system by input clock, bypassing the PLL
- Loss-of-lock reset
- 32/32.768-kHz reference crystal oscillator for the real time clock (RTC) module. Input clock used is programmable within the RTC.

7.1.3 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 RSTOUT asserts until it negates. Refer to Chapter 9, "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.

7.1.3.1 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 22x to 33.75x the input frequency. The user must supply a crystal oscillator that is within the appropriate input frequency range, the crystal manufacturer's recommended external support circuitry, and short signal route from the device to the crystal. In normal mode, the PLL can generate a dithered clock or a non-dithered clock (locked on a single frequency). The dithering deviation, dither modulation frequency, and whether the PLL is modulating or not can be programmed by writing to the PLL registers through the bus interface.

7.1.3.2 Normal PLL Mode with External Reference

Same as Section 7.1.3.1, "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 9, "Chip Configuration Module (CCM)," for details on setting the device for external reference.

7.1.3.3 Input Clock (Limp) Mode

Through the use of RCON, the device may be booted 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 8, "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 3:1 ratio is maintained 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 and FEC are not functional in limp mode.

When switching from LIMP mode to normal functional mode, you must ensure that any peripheral transactions in progress (e.g. Ethernet frame reception/transmission or LCD refresh) are allowed to complete to avoid data loss or corruption.

Limp mode may also be entered and exited from by writing to the MISCCR[LIMP] bit. This is useful because it places the PLL in a state where the multiplication factor (PFMDR) can be altered. Entering limp mode also requires a special procedure with the SDRAM module. As noted above the SDRAM controller is disabled in limp mode, so two critical steps must be followed 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 is shut down.

7.1.3.4 Low-power Mode Operation

This subsection describes the operation of the clock module in low-power and halted modes of operation. Low-power modes are described in Chapter 8, "Power Management." Table 7-1 shows the clock module operation in low-power modes.

Table 7-1. Clock Module Operation in Low-power Modes

Low-power Mode	Clock Operation	Mode Exit
Wait	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Stop	All system clocks disabled	Exit not caused by clock module, but clock sources are re-enabled and normal clocking resumes upon mode exit
Halted	Normal	Exit not caused by clock module

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. There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wakeup recovery time. The PLL can be disabled in stop mode, but requires a wakeup period before it can relock. The oscillator can also be disabled during stop mode, but requires a wakeup 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 8.2.5, "Low-Power Control Register (LPCR)," for more information about operating the PLL in stop mode.

There is also a fast wakeup option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wakeup recovery time but at the risk of sending a potentially unstable clock to the system.

7.2 Memory Map/Register Definition

The PLL module programming model consists of the following registers:

Table 7-2. PLL Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_0000	PLL Output Divider Register (PODR)	8	R/W	0x26	7.2.1/7-6
0xFC0C_0004	PLL Control Register (PCR)	8	R/W	0x00	7.2.2/7-6

Table 7-2. PLL Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_0008	PLL Modulation Divider Register (PMDR)	8	R/W	0x00	7.2.3/7-7
0xFC0C_000C	PLL Feedback Divider Register (PFDR)	8	R/W	0x5A[1]	7.2.4/7-8

[1] With default reset configuration ($\overline{\text{RCON}}$ is negated).

7.2.1 PLL Output Divider Register (PODR)

The PODR register controls the output divider for generating the core and bus clocks. The value of this register is fixed at 0x26 and writing any other value to this register results in unpredictable behaivor.

Address: 0xFC0C_0000 (PODR)   Access:User read/write

	7	6	5	4	3	2	1	0
R								
W	CPUDIV				BUSDIV			
Reset:	0	0	1	0	0	1	1	0
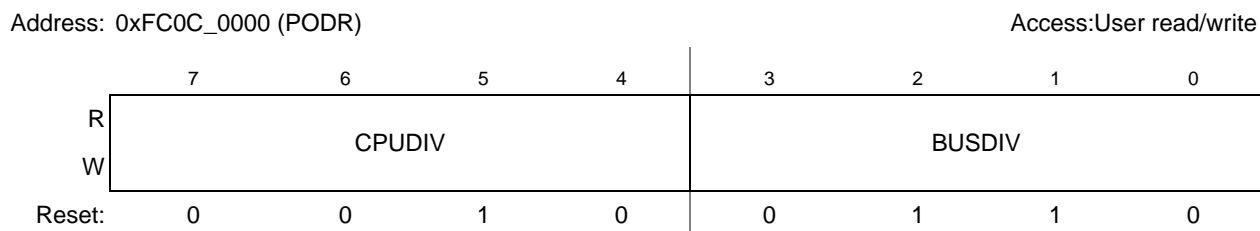
Figure 7-3. PLL Output Divider Register (PODR)

Table 7-3. PODR Field Descriptions

Field	Description
7–4 CPUDIV	Divider for generating the core frequency. See BUSDIV for a table of possible values.
3–0 BUSDIV	Divider for generating the internal bus frequency. <table><tr><th>Value</th><th>Core Clock</th><th>Bus Clock</th></tr><tr><td>0010</td><td>VCO/2</td><td>Reserved</td></tr><tr><td>0110</td><td>Reserved</td><td>VCO/6</td></tr><tr><td>Else</td><td>Reserved</td><td>Reserved</td></tr></table>

7.2.2 PLL Control Register (PCR)

Address: 0xFC0C_0004 (PCR)   Access:User read/write

	7	6	5	4	3	2	1	0
R	DITHEN	0	0	0	0	DITHDEV		
W								
Reset:	0	0	0	0	0	0	0	0
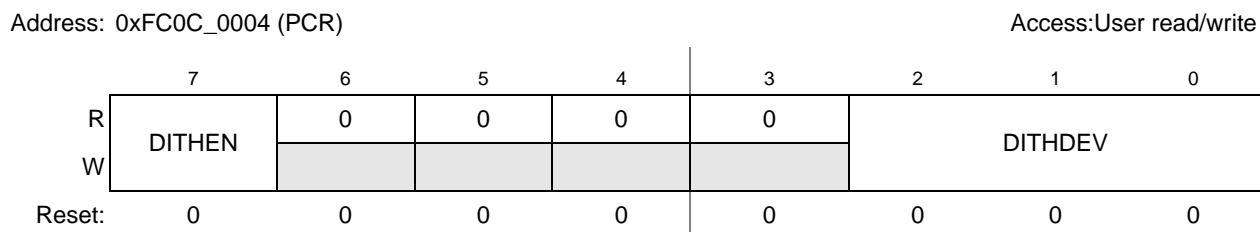
Figure 7-4. PLL Control Register (PCR)

Table 7-4. PCR Field Descriptions

Field	Description
7 DITHEN	Dithering enable bit. 0 Dithering disabled. 1 Dithering enabled.
6–3	Reserved, should be cleared.
2–0 DITHDEV	Dither Deviation. The dither deviation settings are target percentages based on simulation. The actual percentages observed are slightly larger than these targets. See Section 7.3.2, "Dithering Waveform Definition" for more information. Deviation = -0.75% - (DITHDEV $\times$ 0.75%) <table><tr><td>**DITHDEV**</td><td>**Deviation**</td></tr><tr><td>000</td><td>- 0.75%</td></tr><tr><td>001</td><td>- 1.00%</td></tr><tr><td>010</td><td>- 1.25%</td></tr><tr><td>011</td><td>- 1.50%</td></tr><tr><td>100</td><td>- 1.75%</td></tr><tr><td>101</td><td>- 2.00%</td></tr><tr><td>110</td><td>- 2.25%</td></tr><tr><td>111</td><td>- 2.50%</td></tr></table> **Note:** This field should only be written when dithering mode is disabled (PCR[DITHEN] = 0). Else, unpredictable PLL operation results.

7.2.3 PLL Modulation Divider Register (PMDR)

Address: 0xFC0C_0008 (PMDR)   Access: User read/write

	7	6	5	4	3	2	1	0
R W				MODDIV				
Reset:	0	0	0	0	0	0	0	0
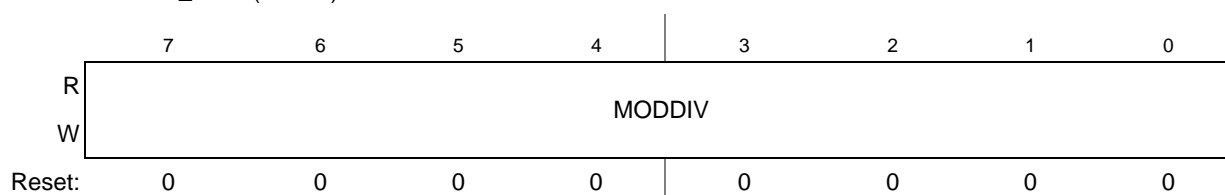
Figure 7-5. PLL Modulation Divider Register (PMDR)

Table 7-5. PMDR Field Descriptions

Field	Description
7–0 MODDIV	Dither modulation divider. Dither Modulation Frequency = Input Frequency / (MODDIV × 32) A dither modulation frequency greater than 105 kHz or less than 9.95 kHz is invalid. For example, for a 16 MHz input frequency, MODDIV may be programmed between 5 (100 kHz) and 50 (10 kHz). Programming MODDIV outside the specified range results in unpredictable PLL operation. **Note:** This field should only be written when dithering mode is disabled (PCR[DITHEN] = 0). Else, unpredictable PLL operation results.

7.2.4 PLL Feedback Divider Register (PFDR)

Address: 0xFC0C_000C (PFDR)   Access: User read/write

	7	6	5	4	3	2	1	0
R					MFD			
W								
Reset:	See Note	See Note	See Note	See Note	See Note	See Note	See Note	See Note

Note: Reset value determined by reset configuration. See Chapter 9, "Chip Configuration Module (CCM)," for more information. For the default reset configuration ($\overline{RCON}$ negated), the reset value is 0x5A. If $\overline{RCON}$ and D1 is asserted at reset, the reset value of PFDR is 0x78.

Figure 7-6. PLL Feedback Divider Register (PFDR)

Table 7-6. PFDR Field Descriptions

Field	Description
7–0 MFD	The MFD bits control the value of the divider in the PLL feedback loop. The value specified by the MFD bits establish the multiplication factor applied to the reference frequency. See Section 7.3.3, "PLL Frequency Multiplication Factor Select" for more details. 0x58 88 0x59 89 0x5A 90 ... 0x86 134 0x87 135 Else Reserved **Note:** The MFD bits may only be written when the device is in limp mode (MISCCR[LIMP] = 1).

7.3 Functional Description

This subsection provides a functional description of the clock module.

7.3.1 PLL Dithered and Non-Dithered Operation

The PLL is capable of generating output clocks with a frequency that modulates in a triangular waveform with a specified percentage frequency deviation and a specified dither modulation frequency. This modulation of the output clock is called dithered operation. When the PLL operates at a fixed frequency,

this operation is known as non-dithered operation. The selection of dithered or non-dithered operation is controlled by the PCR[DITHEN] bit. The percent frequency deviation and dither modulation frequency are also controlled by the PCR and PMDR registers, whose operation is described in Section 7.3.2, "Dithering Waveform Definition."

After reset, dithering is disabled. After the PLL has locked (indicated by the MISCCR[PLLLOCK] bit described in Section 9.3.4, "Miscellaneous Control Register (MISCCR)"), the PLL may be changed from non-dithered operation to dithered operation by writing to the PCR. After the PCR[DITHEN] bit has been set, the PLL synchronizes the new value with the VCO clock domain. Then the transition from non-dithered operation to dithered operation takes place such that the PLL output clocks remain glitch-free. However, the dithering waveform and deviation percentages are not guaranteed to meet specifications until two modulation periods have passed after the time of the write to the PCR register. During the transition the frequency of the PLL output clocks does not exceed 10% of the respective non-dithered frequency.

The PLL may also be changed from dithered operation to non-dithered operation by clearing the PCR[DITHEN] bit. After this occurs, the PLL synchronizes the new value with the VCO clock domain. Then, the transition from dithered operation to non-dithered operation takes place such that PLL output clocks remain glitch-free. However, the frequency of the PLL output clocks are not guaranteed to be completely stable until one modulation period has passed after the time of the write to the PCR. During the transition the frequency of the PLL output clocks does not exceed 10% of the respective non-dithered frequency.

Because the transition between dithered and non-dithered operation (and vice-versa) takes a period of time to change, the PCR may not be written back-to-back without waiting two modulation periods between writes.

NOTE

Failure to wait two modulation periods between writes to the PCR results in unpredictable PLL operation.

7.3.2 Dithering Waveform Definition

The dithering waveform created by the changes in the frequency of the PLL output clocks is defined by the percent frequency deviation and dither modulation frequency. The definitions of the percent frequency deviation (or dithering deviation) and the modulation period (which is the inverse of the dither modulation frequency) are shown in Figure 7-7. The dithering deviation is controlled by PCR[DITHDEV] field, while the dither modulation frequency is controlled by the PMDR[MODDIV] field.
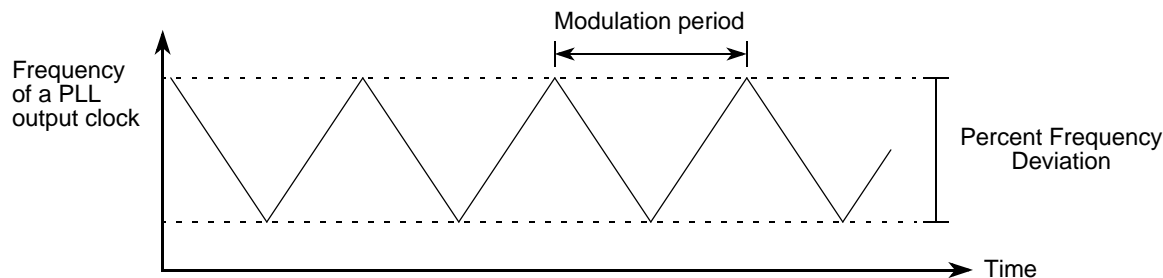
Figure 7-7. Ideal Triangular Dithering Waveform

After reset, the PCR and PMDR registers are cleared, disabling dithering. After reset has been de-asserted and the PLL has locked (indicated by the MISCCR[PLLLOCK] bit), the dithering waveform definition may be changed by writing to the PCR[DITHDEV] field and the PMDR register. However, the PCR[DITHDEV] field and PMDR register may only be written to when the PLL is in non-dithered operation (i.e. the PCR[DITHEN] bit must be cleared).

NOTE

Writing to the PCR[DITHDEV] field or the PMDR during dithering operation results in unpredictable PLL operation.

The dither deviation can be programmed to be between -0.75% of the non-dithered frequency up to -2.50% of the non-dithered frequency in steps of 0.25%.

NOTE

The dither deviation settings in the PCR[DITHDEV] field are target percentages based on simulation. The actual percentages achieved may be numerically different than the percentages listed in the specification; however, the actual percentages achieved are in proportion to each other and are stable within ±10% across process, voltage, and temperature conditions.

The dither modulation frequency can be programmed to be between approximately 10kHz and 100kHz. Because the dither modulation frequency is determined as a division of the input frequency, the dither modulation frequency is given by the following equation:

$$\text{Dither Modulation Frequency} = \frac{\text{Input Frequency}}{\text{PMDR[MODDIV]} \times 32} \qquad \textit{Eqn. 7-1}$$

NOTE

PMDR[MODDIV] field values that result in a dither modulation frequency greater than 105kHz or less than 9.95kHz are invalid and result in unpredictable PLL operation.

7.3.3   PLL Frequency Multiplication Factor Select

The frequency multiplication factor of the PLL is defined by the feedback divider in the following equation:

$$f_{sys} = f_{ref} \times \left(\frac{\text{PFDR}}{4 \times \text{PODR[CPUDIV]}} \right) \qquad \textit{Eqn. 7-2}$$

where f_{sys} is the core frequency. The allowable range of values for the PFDR is 88 to 135, resulting in a frequency multiplication factor range of 11 to 16.88 times the input reference frequency (typically 16 MHz).

The PFDR can only be modified while the device is in limp mode, which is entered by setting the MISCCR[LIMP] bit. After the PFDR register has been changed, re-enter normal mode by clearing the MISCCR[LIMP] bit. The PLL then begins to acquire lock accordingly on the new frequency.

7.3.4 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/bus frequency of 180/60 MHz. This default mode can be overridden by asserting the $\overline{\text{RCON}}$ pin. See Chapter 9, "Chip Configuration Module (CCM)," for more information on overriding the default configuration during reset.

Table 7-7 shows the clock-out frequency to clock-in frequency relationships for the possible system clock modes. Refer to Section 7.1.3, "Modes of Operation" for details on each mode.

Table 7-7. Clock Out and Clock In Relationships

System Clock Mode	PLL Options[1]	Cross-Reference
Normal PLL clock mode	$f_{sys} = f_{ref} \times \left(\dfrac{\text{PFDR}}{8} \right)$	Section 7.1.3.1, "Normal PLL Mode with Crystal Reference" and Section 7.1.3.2, "Normal PLL Mode with External Reference"
Limp mode	$f_{sys} = \dfrac{f_{ref}}{2^{\text{MISCCR[LPDIV]}}}$	Section 7.1.3.3, "Input Clock (Limp) Mode"

[1] f_{ref} = input reference frequency = 16 MHz
   PFDR ranges from 88 to 135
   MISCCR[LPDIV] ranges from 0 to 15

7.3.5 Clock Operation During Reset

This section describes the reset operation of the PLL. Power-on reset and normal reset are described.

7.3.5.1 Power-On Reset (POR)

After V_{DDPLL} and the input clock are within specification, the PLL is held in reset for at least 10 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 $\overline{\text{RESET}}$ is de-asserted, PLL output clocks are generated; however, until the MISCCR[PLLLOCK] bit is set the PLL output clock frequencies are not stable and not within specification. The MISCCR[PLLLOCK] bit is set after $\overline{\text{RESET}}$ has negated for a minimum of 1 ms. When this bit is set, the PLL is in frequency lock.

7.3.5.2 External Reset

When $\overline{\text{RESET}}$ is asserted, the PLL input clock is output to the device and the PLL does not begin acquiring lock until $\overline{\text{RESET}}$ is negated. The MISCCR[PLLLOCK] bit is cleared and remains cleared while the PLL is acquiring lock. This bit is set after the PLL lock period of 1 ms has passed.

<div align="center">

CAUTION

</div>

> When running in an unlocked state, the clocks generated by the PLL are not guaranteed to be stable and may exceed the maximum specified frequency of the device.

Chapter 8
Power Management

8.1   Introduction

This chapter explains the low-power operation of the device.

8.1.1   Features

The following features support low-power operation:

- Four modes of operation: 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

8.2   Memory Map/Register Definition

The power management programming model consists of registers from the SCM and CCM memory space, as shown below:

Table 8-1. 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	8.2.1/8-2
0xFC04_002C	Peripheral Power Management Set Register 0 (PPMSR0)	8	W	0x00	8.2.2/8-3
0xFC04_002D	Peripheral Power Management Clear Register 0 (PPMCR0)	8	W	0x00	8.2.3/8-4
0xFC04_002E	Peripheral Power Management Set Register 1 (PPMSR1)	8	W	0x00	8.2.2/8-3
0xFC04_002F	Peripheral Power Management Clear Register 1 (PPMCR1)	8	W	0x00	8.2.3/8-4
0xFC04_0030	Peripheral Power Management High Register 0 (PPMHR0)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC04_0034	Peripheral Power Management Low Register 0 (PPMLR0)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC04_0038	Peripheral Power Management High Register 1 (PPMHR1)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC0A_0007	Low-Power Control Register (LPCR)	8	R/W	0x00	8.2.5/8-7

Table 8-1. Power Management Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0010	Miscellaneous Control Register (MISCCR)[2]	16	R/W	See Section	9.3.4/9-5
0xFC0A_0012	Clock Divider Register (CDR)[2]	16	R/W	0x0001	9.3.5/9-6

[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 9, "Chip Configuration Module (CCM)."

8.2.1 Wake-up Control Register

Implementation of low-power stop mode and exit from a low-power mode via an interrupt require communication between the core and logic associated with the interrupt controller. The WCR enables entry into low-power modes, and includes the setting of the interrupt level needed to exit a low-power mode.

NOTE

The setting of the low-power mode select field, LPCR[LPMD], determines which low-power mode the device enters when a STOP instruction is issued.

The following sequence of operations is 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 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 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	0	0	0	PRILVL		
W								
Reset:	0	0	0	0	0	0	0	0
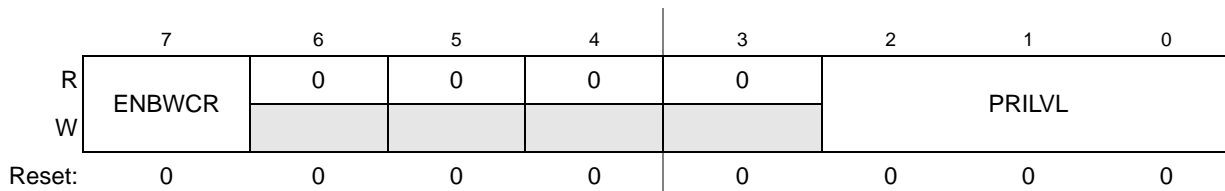
Figure 8-1. Wake-up Control Register (WCR)

Table 8-2. WCR Field Descriptions

Field	Description
7 ENBWCR	Enable low-power mode entry. The mode entered is specified in LPCR[LPMD]. 0 Low-power mode entry is disabled 1 Low-power mode entry is enabled.
6–3	Reserved, should be cleared.
2–0 PRILVL	Exit low-power mode interrupt priority level. This field defines the interrupt priority level needed 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>11x</td><td>Interrupt request level [7] exits low-power mode</td></tr></table>

8.2.2 Peripheral Power Management Set Registers (PPMSR0 & PPMSR1)

The PPMSR registers provide 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 the need 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 be set, disabling all peripheral module clocks. Reads of these registers return all zeroes.

Address: 0xFC04_002C (PPMSR0)   Access: Supervisor Write-only
         0xFC04_002E (PPMSR1)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAMCD	SMCD					
Reset:	0	0	0	0	0	0	0	0
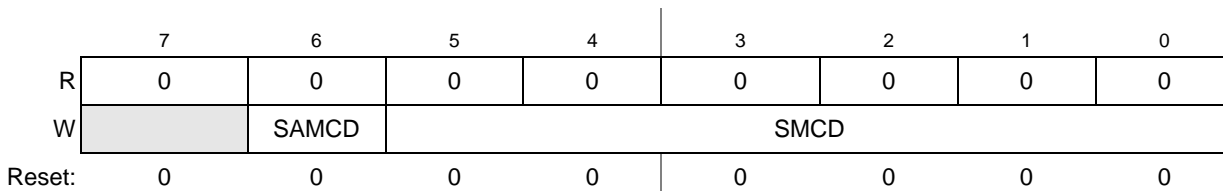
Figure 8-2. Peripheral Power Management Set Registers (PPMSR*n*)

Table 8-3. PPMSR*n* Field Descriptions

Field	Description
7	Reserved, should be cleared.

Table 8-3. PPMSR*n* Field Descriptions (continued)

Field	Description
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.

8.2.3 Peripheral Power Management Clear Registers (PPMCR0 & PPMCR1)

The PPMCR registers provide a simple mechanism to clear a given bit in the PPMHR & PPMLR registers, enabling the clock for a given peripheral module without the need 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 contents of the PPMR to be clear, enabling all peripheral module clocks. Reads of these registers return all zeroes.

Address: 0xFC04_002D (PPMCR0)   Access: Supervisor Write-only
0xFC04_002F (PPMCR1)

	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
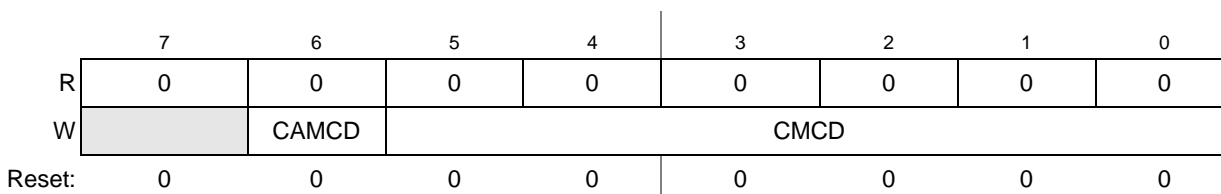
Figure 8-3. Peripheral Power Management Clear Registers (PPMCR*n*)

Table 8-4. PPMCR*n* Field Descriptions

Field	Description
7	Reserved, should 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.

8.2.4 Peripheral Power Management Registers (PPMHR0, PPMHR1, & 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 of these address spaces 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.

The individual bits of the PPMR can be modified using a read-modify-write to this register directly or indirectly through writes to the PPMSR and PPMCR registers to set/clear individual bits.

Address: 0xFC04_0038 (PPMHR1)   Access: Supervisor 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	CD34	CD33	CD32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-4. Peripheral Power Management High Register (PPMHR1)

Table 8-5. PPMHR1[CD*n*] Assignments

Slot Number	CD*n*	Peripheral
32	CD32	MDHA
33	CD33	SKHA
34	CD34	Random Number Generator

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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CD48
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	CD47	CD46	CD45	CD44	CD43	CD42	CD41	CD40	0	CD38	CD37	CD36	CD35	CD34	CD33	CD32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-5. Peripheral Power Management High Register (PPMHR0)

Table 8-6. 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

Table 8-6. PPMHR0[CD*n*] Assignments (continued)

Slot Number	CD*n*	Peripheral
36	CD36	PWM
37	CD37	Edge Port
38	CD38	On-chip Watchdog Timer
40	CD40	CCM, Reset Controller, Power Management
41	CD41	GPIO Module
42	CD42	Real Time Clock
43	CD43	LCD Controller
44	CD44	USB On-the-Go
45	CD45	USB Host
46	CD46	SDRAM Controller
47	CD47	SSI
48	CD48	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	0	0	0	CD12	0	0	0	CD8	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
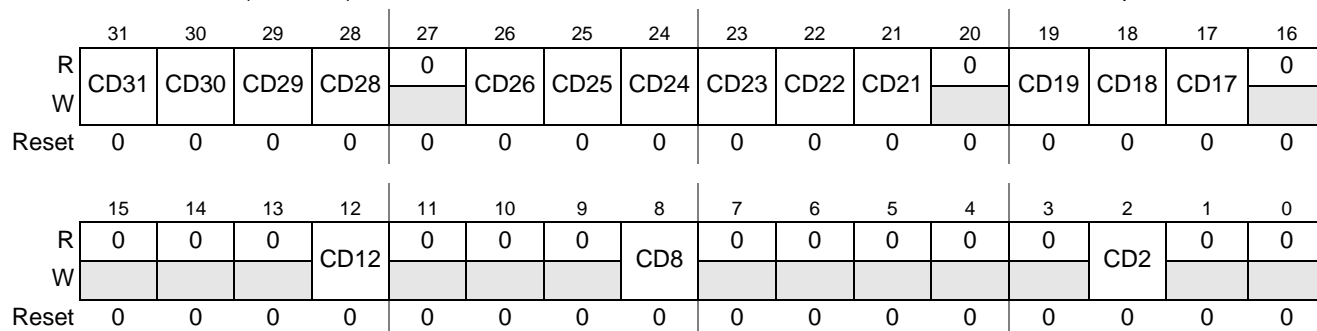
Figure 8-6. Peripheral Power Management Low Registers (PPMLR0)

Table 8-7. PPMLR0[CD*n*] Assignments

Slot Number	CD*n*	Peripheral
2	CD2	FlexBus
8	CD8	FlexCAN
12	CD12	FEC
17	CD17	eDMA Controller
18	CD18	Interrupt Controller 0
19	CD19	Interrupt Controller 1
21	CD21	IACK
22	CD22	I^2C
23	CD23	QSPI

Table 8-7. PPMLR0[CD*n*] Assignments (continued)

Slot Number	CD*n*	Peripheral
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 8-8. PPMHR & 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

Extreme caution should be taken by the customer when setting PPMR[CD40] to disable clocking of the CCM, reset controller, and power management modules. This may disable logic to reset the chip, disable the external bus monitor, and other logic contained within these blocks.

8.2.5 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	LPMD		FWKUP	STPMD		0	0	0
W								
Reset:	0	0	0	0	0	0	0	0
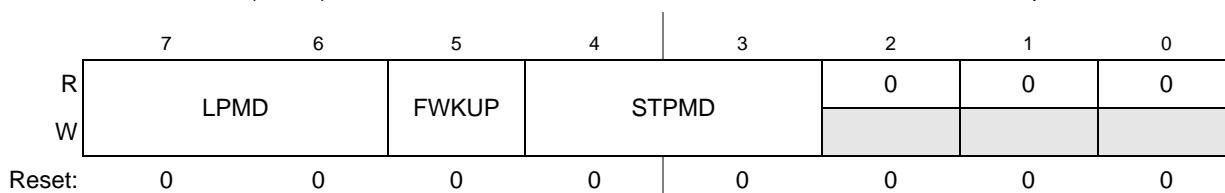
Figure 8-7. Low-Power Control Register (LPCR)

Table 8-9. LPCR Field Descriptions

Field	Description
7–6 LPMD	Low-power mode select. Used to select the low-power mode the chip enters after the ColdFire core executes the STOP instruction. These bits must be written prior to instruction execution for them to take effect. The LPMD bits are readable and writable in all modes. 00 Run 01 Doze 10 Wait 11 Stop **Note:** If LPCR[LPMD] is cleared, the device stops executing code upon issue of a STOP instruction. However, no clocks are disabled.
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 the MISCCR[PLLLOCK] bit. FWKUP is not effective in limp mode because the PLL never locks in this mode. 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 as shown below. | 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, should be cleared.

8.3 Functional Description

The functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes are discussed in this section.

8.3.1 Peripheral Shut Down

All peripherals, except for the SCM and crossbar switch, may have their input clocks individually removed by software to reduce power consumption. See Section 8.2.4, "Peripheral Power Management Registers (PPMHR0, PPMHR1, & PPMLR0)" for more information. A peripheral may be disabled at any time and remains disabled during any low-power mode of operation.

8.3.2 Limp mode

The device may also be booted 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, 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 also be entered and exited from by writing to the MISCCR[LIMP] bit.

While in this mode a 3:1 ratio is maintained 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, both USB modules, and FEC are not functional in limp mode.

8.3.3 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 LPCR[LPMD]. 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 which is 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.

8.3.3.1 Run Mode

Run mode is the normal system operating mode. Current consumption in this mode is related directly to the system clock frequency.

8.3.3.2 Wait Mode

Wait mode is intended to be used 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, which cause the CPU to exit from wait mode.

8.3.3.3 Doze Mode

Doze mode affects the CPU in the same manner as wait mode, except that some peripherals define individual operational characteristics in doze mode. Peripherals which continue to run and have the

capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Peripherals that are stopped restart operation on exit from doze mode as defined for each peripheral.

8.3.3.4 Stop Mode

Stop mode affects the CPU in the same manner 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.

The following subsections specify the operation of each module while in and when exiting low-power modes.

NOTE

Entering stop mode disables the SDRAMC including the refresh counter. If SDRAM is used, then code is required to ensure proper entry and exit from stop mode. See Chapter 18, "SDRAM Controller (SDRAMC)," for more information.

8.3.4 Peripheral Behavior in Low-Power Modes

8.3.4.1 ColdFire Core

The ColdFire core is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

8.3.4.2 Internal SRAM

The SRAM is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

8.3.4.3 Clock Module

In wait and doze modes, the clocks to the CPU and SRAM is stopped and the system clocks to the peripherals are enabled. 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 8.2.4, "Peripheral Power Management Registers (PPMHR0, PPMHR1, & PPMLR0)"). In stop mode, all clocks to the system are stopped.

There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wakeup recovery time. The PLL can be disabled in stop mode, but requires a wakeup period before it can relock. The oscillator can also be disabled during stop mode, but requires a wakeup 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 8.2.5, "Low-Power Control Register (LPCR)," for more information about operating the PLL in stop mode.

There is also a fast wakeup option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wakeup recovery time but at the risk of sending a potentially unstable clock to the system. This is also explained in Section 8.2.5, "Low-Power Control Register (LPCR)."

8.3.4.4 Chip Configuration Module

The chip configuration module is unaffected by entry into a low-power mode. If low-power mode is exited by a reset, chip configuration may be executed if configured to do so.

8.3.4.5 Reset Controller

A power-on reset (POR) always causes a chip 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 exit any low-power modes.

In stop mode, the $\overline{\text{RESET}}$ pin synchronization is disabled 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 or on-chip watchdog timer remains enabled during wait or doze modes, then a watchdog timer timeout may generate a reset to exit these low-power modes.

When the CPU is inactive, a software reset cannot be generated to exit any low-power mode.

8.3.4.6 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 cause a reset of 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 8.3.3, "Low-Power Modes" for the core watchdog interrupt to bring the part out of low-power mode.

8.3.4.7 Cross-Bar Switch

8.3.4.8 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.

8.3.4.9 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 are stopped.

An interrupt request causes the CPU 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.

8.3.4.10 Edge Port

In wait and doze modes, the edge port continues to operate normally and may be configured to generate interrupts (an edge transition or low level on an external pin) to exit the low-power modes.

In stop mode, there is no system clock available to perform the edge detect function. Thus, 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.

8.3.4.11 eDMA Controller

In wait and doze modes, the eDMA controller is capable of bringing the device out of a low-power mode by generating an interrupt upon completion of a transfer or an error condition. The completion of transfer interrupt is generated when DMA interrupts are enabled by the setting of a EDMA_INTR[INTn] bit, and an interrupt is generated when the TCDn[DONE] bit is set. The interrupt upon error condition is generated when the EDMA_EEIR[EEIn] bit is set, and an interrupt is generated when any of the EDMA_ESR bits becomes set.

The eDMA controller is stopped in stop mode and thus cannot cause an exit from this low-power mode.

8.3.4.12 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.

8.3.4.13 SDRAM Controller (SDRAMC)

SDRAM controller operation is unaffected by the wait or doze modes; however, the SDRAMC is disabled by stop mode. Because all clocks to the SDRAMC are disabled by stop mode, the SDRAMC does not generate refresh cycles.

To prevent loss of data the 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 are used to maintain the integrity of the data stored in the SDRAM.

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 while 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.

8.3.4.14 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. During this mode, the FEC clocks are shut down. Coming out of stop mode returns the FEC to operation from the state prior to stop mode entry.

8.3.4.15 FlexCAN

When enabled, the FlexCAN module is capable of generating interrupts and bringing the device out of a low-power mode. The module has 18 interrupt sources (16 sources due to message buffers and 2 sources due to bus-off and error).

When setting stop mode in the FlexCAN (by setting the CANMCR[MDIS] bit), the FlexCAN checks for the CAN bus to be idle or waits for the third bit of intermission and checks to see if it is recessive. When this condition exists, the FlexCAN waits for all internal activity other than in the CAN bus interface to complete and then the following occurs:

- The FlexCAN shuts down its clocks, stopping most of the internal circuits, to achieve maximum possible power saving.
- The internal bus interface logic continues operation, enabling CPU to access the CANMCR register.
- The FlexCAN ignores its Rx input pin, and drives its Tx pins as recessive.
- FlexCAN loses synchronization with the CAN bus, and the CANMCR[STOP_ACK, NOT_RDY] bits are set.

Exiting stop mode is done in one of the following ways:

- Reset the FlexCAN (by hard reset or by asserting the CANMCR[SOFT_RST] bit).
- Clearing the CANMCR[MDIS] bit.

Recommendations for, and features of, FlexCAN's stop mode operation are as follows:

- Upon stop mode entry, the FlexCAN tries to receive the frame that caused it to wake; that is, it assumes that the dominant bit detected is a start-of-frame bit. It does not arbitrate for the CAN bus then.
- Before asserting stop mode, the CPU should disable all interrupts in the FlexCAN, otherwise it may be interrupted while in stop mode upon a non-wake-up condition.
- If stop mode is asserted while the FlexCAN is BUSOFF (see error and status register), then the FlexCAN enters stop mode and stops counting the synchronization sequence; it continues this count after stop mode is exited.

- If halt mode is active at the time the MDIS bit is set, then the FlexCAN assumes that halt mode should be exited; hence it tries to synchronize to the CAN bus (11 consecutive recessive bits), and only then does it search for the correct conditions to stop.
- Trying to stop the FlexCAN immediately after reset is allowed only after basic initialization has been performed.

8.3.4.16 On-chip Watchdog Timer

In stop mode (or in wait/doze mode, if so programmed in the WCR register), the watchdog ceases operation and freezes at the current value. When exiting these modes, the watchdog resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the watchdog may generate a reset to exit the low-power modes.

8.3.4.17 PWM Module

The PWM module is user programmable as to how it behaves when the device enters wait mode (PWMCTL[PSWAI]) and debug mode (PWMCTL[PFRZ]). If either of these bits are set, the PWM input clock to the prescalar is disabled during the respective low- power mode.

In stop mode the input clock is disabled and PWM generation is halted.

8.3.4.18 Programmable Interrupt Timers (PIT0–3)

In stop mode (or in doze mode, if so programmed in the PCSRn 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.

8.3.4.19 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 be generated 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 is issued upon a captured input. In reference compare mode, where the output reference request interrupt enable (ORRI) bit of DTMR is set and the DTXMR[DMAEN] bit is cleared, an interrupt is issued when the timer counter reaches the reference value.

DMA timer operation is disabled in stop mode. Upon exiting stop mode, the timer resumes operation unless stop mode was exited by reset.

8.3.4.20 Queued Serial Peripheral Interface (QSPI)

In wait and doze modes, the QSPI module is unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the QSPI stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the QSPI clocks are shut down. Coming out of stop mode returns the QSPI to operation from the state prior to stop mode entry.

8.3.4.21 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 are shut down. Coming out of stop mode returns the UARTs to operation from the state prior to stop mode entry.

8.3.4.22 I^{2}C Module

When the I^{2}C Module is enabled by the setting of the I2CR[IEN] bit and when 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.

8.3.4.23 BDM

Entering halt (debug) mode via the BDM port (by asserting the external $\overline{\text{BKPT}}$ pin) causes the CPU to exit any low-power mode.

8.3.4.24 JTAG

The JTAG (Joint Test Action Group) controller logic is clocked using the TCLK input and is not affected by the system clock. The JTAG cannot generate an event to cause the CPU to exit any low-power mode. Toggling TCLK during any low-power mode increases the system current consumption.

8.3.5 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 8-10. 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 LPCR[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 8-10. 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	Enabled	Reset
System Control Module	Enabled	Reset	Enabled	Reset	Stopped	N/A
GPIO Module	Enabled	N/A	Enabled	N/A	Enabled	N/A
Interrupt controller	Enabled	Interrupt	Enabled	Interrupt	Enabled	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 Host	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
USB OTG	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
LCD Controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
FlexCAN	Enabled	Reset	Enabled	Reset	Stopped	N/A
SSI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
Real Time Clock	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
On-chip Watchdog Timer	Program	Reset	Program	Reset	Stopped	N/A
PWM	Program	N/A	Program	N/A	Stopped	N/A
Programmable Interrupt Timers	Enabled	Interrupt	Program	Interrupt	Stopped	N/A
DMA Timers	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
QSPI	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
Cryptography Modules (RNG, SKHA, MDHA)	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
JTAG	Enabled	N/A	Enabled	N/A	Enabled	N/A
BDM[2]	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 BDM logic is clocked by a separate TCLK clock. 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 9
Chip Configuration Module (CCM)

9.1 Introduction

The chip configuration module (CCM) controls the chip configuration for the device.

9.1.1 Block Diagram
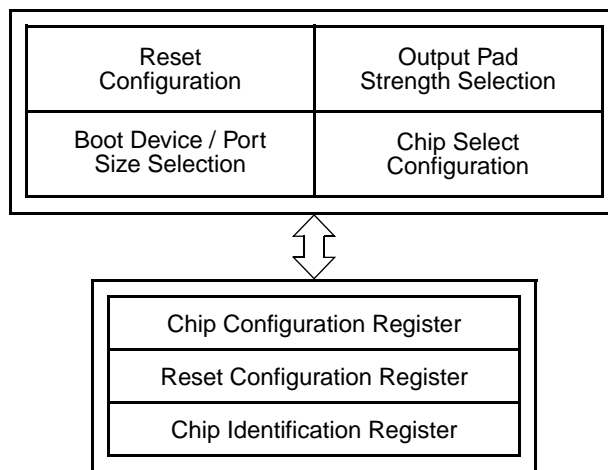
Figure 9-1. Chip Configuration Module Block Diagram

9.1.2 Features

The CCM performs the following operations:

- Selects the chip operating mode
- Selects external clock or phase-lock loop (PLL) mode with internal or external reference
- Selects output pad drive strength
- Selects boot device and data port size
- Selects bus monitor configuration
- Selects low-power configuration

9.1.3 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 six chip selects can be programmed to select and control external devices and to provide bus cycle termination.

9.2 External Signal Descriptions

Table 9-1 provides an overview of the CCM signals.

Table 9-1. Signal Properties

Name	Function	Reset State
$\overline{\text{RCON}}$	Reset configuration select	Internal weak pull-up device
D[9:86:1]	Reset configuration override pins	—

9.2.1 $\overline{\text{RCON}}$

If the external $\overline{\text{RCON}}$ pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins (see Section 9.4, "Functional Description"). The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

9.2.2 D[9:86:1] (Reset Configuration Override)

If the external $\overline{\text{RCON}}$ pin is asserted during reset, then the states of these data pins during reset determine the chip mode of operation, boot device, clock mode, and certain module configurations after reset.

NOTE
It is recommended that the logic levels for reset configuration on D[9:86:1] be actively driven when $\overline{\text{RCON}}$ is used. The rest of the data bus should be allowed to float or be pulled high.

9.3 Memory Map/Register Definition

The CCM programming model consists of the registers listed in the below table.

Table 9-2. 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	9.3.1/9-3
0xFC0A_0008	Reset Configuration Register (RCON)	16	R	0x0001	9.3.2/9-4
0xFC0A_000A	Chip Identification Register (CIR)	16	R	See Section	9.3.3/9-4
0xFC0A_0010	Miscellaneous Control Register (MISCCR)	16	R/W	See Section	9.3.4/9-5
0xFC0A_0012	Clock Divider Register (CDR)	16	R/W	0x0001	9.3.5/9-6

Table 9-2. CCM Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0014	USB Host Controller Status Register (UHCSR)	16	R/W	0x0000	9.3.6/9-7
0xFC0A_0016	USB On-the-Go Controller Status Register (UOCSR)	16	R/W	0x0010	9.3.7/9-7

[1] User access to supervisor only address locations have no effect and result in a bus error.

9.3.1 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 may be overridden during reset configuration using the external RCON and D[15:0] pins. (See Figure 9-3 for the RCON register definition.)

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	CSC		0	LIMP	LOAD	BOOTPS		OSC MODE	PLL MODE	1
W																
Reset	0	0	0	0	0	0					See Note					1

Note: Reset value depends upon chosen reset configuration. Default reset value ($\overline{\text{RCON}}$ is not asserted) is 0x0001.

Figure 9-2. Chip Configuration Register (CCR)

Table 9-3. CCR Field Descriptions

Field	Description
15–10	Reserved, should be cleared.
9–8 CSC	Chip select configuration field. Reflects the chosen chip select configuration. 00 A[23:22] = A[23:22] 01 Reserved 10 A[23] = $\overline{\text{FB\_CS5}}$ and A[22] = A[22] 11 A[23:22] = $\overline{\text{FB\_CS}}$[5:4]
7	Reserved, should be cleared.
6 LIMP	Limp mode bit. 0 Normal operation; PLL drives internal clocks. 1 Limp mode; low-power clock divider drives internal clocks.
5 LOAD	Pad driver load bit. Reflects the chosen pad driver strength for those pins with drive strength control and the chosen pad slew rate for those pins with slew rate control. 0 Low drive strength, low slew rate 1 High drive strength, high slew rate
4–3 BOOTPS	Boot port size field. Indicates the selection for the boot port size. 00 32 bits 01 16 bits 10 8 bits 11 32 bits

Table 9-3. CCR Field Descriptions (continued)

Field	Description
2 OSCMODE	Oscillator clock mode bit. 0 Crystal oscillator mode 1 Oscillator bypass mode
1 PLLMODE	PLL clock mode 0 180/60 MHz operation 1 240/80 MHz operation
0	Reserved, should be set.

9.3.2 Reset Configuration Register (RCON)

At reset, RCON determines the default operation of certain chip functions. All default functions defined by the RCON values can only be overridden during reset configuration (see) if the external $\overline{\text{RCON}}$ pin is asserted. RCON is a read-only register and contains the same fields as the CCR register. See for field descriptions.

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	CSC		0	LIMP	LOAD	BOOTPS		OSC MODE	PLL MODE	1
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
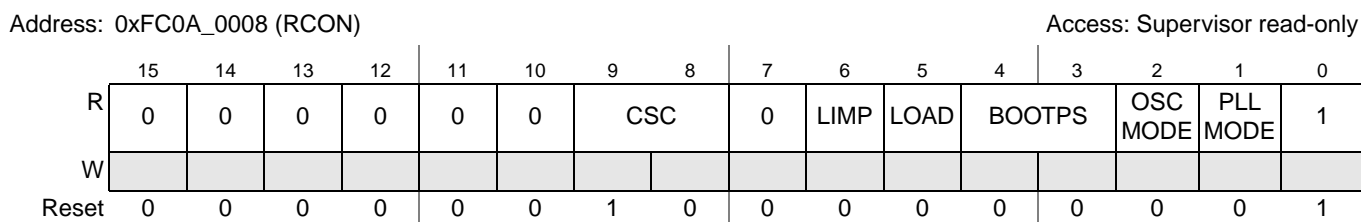
Figure 9-3. Reset Configuration Register (RCON)

9.3.3 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					
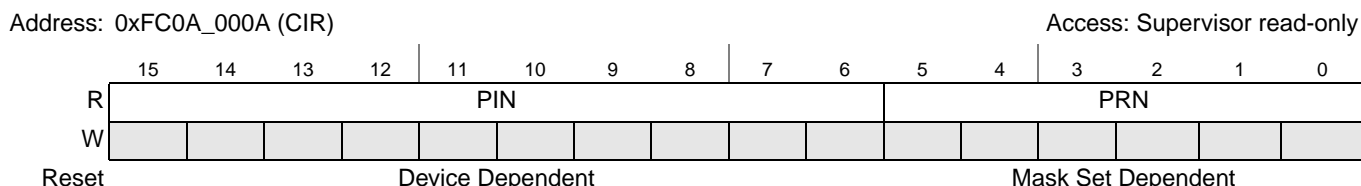
Figure 9-4. Chip Identification Register (CIR)

Table 9-4. CIR Field Description

Field	Description
15–6 PIN	Part identification number. Contains a unique identification number for the device. 0x054 MCF5329 0x058 MCF53281 0x059 MCF5328 0x061 MCF5327
5–0 PRN	Part revision number. This number is increased by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order.

9.3.4 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	PLL LOCK	LIMP	0	0	0	LCD CHEN	SSI PUE	SSI PUS	TIM DMA	SSI SRC	0	0	USB DIV	USB SRC
W																
Reset	0	0	1	0[1]	0	0	0	0	1	1	1	1	0	0	0[2]	1

[1] Reset value depends on RCON RLIMP selection; 0 for PLL operation, 1 for low-power divider operation.

[2] Reset value depends on RCON PLLMODE selection; 0 for 180 MHz operation, 1 for 240 MHz operation.

Figure 9-5. Miscellaneous Control Register (MISCCR)

Table 9-5. MISCCR Field Description

Field	Description
15–14	Reserved, should be cleared.
13 PLLLOCK	PLL lock status. Indicates if the PLL is locked. The PLLLOCK bit is read-only. Writing to PLLLOCK has no effect. 0 PLL is not locked. 1 PLL is locked.
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.
11–9	Reserved, should be cleared.
8 LCDCHEN	LCDC internal clock enable. Selects whether the internal clock input to the LCD controller is enabled. 0 LCDC internal clock disabled 1 LCDC internal clock enabled
7 SSIPUE	SSI RXD/TXD pull enable. Enables the internal weak pull cells on any external pin where the SSI receive data (RXD) function or SSI transmit data (TXD) function is available. The affected pins include SSI_RXD, SSI_TXD, U1RXD, and U1TXD. 0 SSI data pin weak pull cells disabled. 1 SSI data pin weak pull cells enabled. **Note:** The SSIPUE bit only enables the pull cells when the SSI RXD and TXD functions are currently selected for the affected pins. See the Chapter 13, "General Purpose I/O Module," 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.

Table 9-5. MISCCR Field Description (continued)

Field	Description
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 theChapter 16, "Enhanced Direct Memory Access (eDMA)," for more details on the DMA controller. 0 SSI RX0, SSI RX1, SSI TX0, 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_CLK pin as the source of the SSI baud clock. 0 SSI_CLK pin directly drives SSI baud clock. 1 PLL drives SSI baud clock with fractionally divided CPU clock.
3–2	Reserved, should be cleared.
1 USBDIV	USB clock divisor. Indicates the divide value needed to generate the 60 MHz serial clock source to the USB modules. 0 60 MHz USB clock obtained though divide-by-3 (CPU operating at 180 MHz). 1 60 MHz USB clock obtained though divide-by-4 (CPU operating at 240 MHz).
0 USBSRC	USB clock source. Selects between the PLL and the external USB_CLKIN external pin as the clock source for the serial interfaces of the USB modules. 0 USB_CLKIN pin drives USB serial interface clocks. 1 PLL drives USB serial interface clocks.

9.3.5 Clock Divider Register

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				0	0	SSIDIV					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
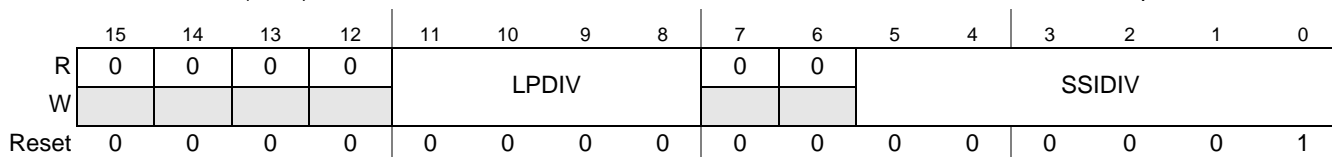
Figure 9-6. Clock Divider Register (CDR)

Table 9-6. CDR Field Description

Field	Description
15–12	Reserved, should be cleared.
11–8 LPDIV	Low power clock divider. Specifies the divide value used to produce the system clocks during LIMP mode. A 3:1 ratio is maintained between the core and the internal bus.This field is used only when MISCCR[LIMP] bit is set. $$\text{System Clocks} = \frac{f_{vco}}{2^{LPDIV}}$$ *Eqn. 9-1* **Note:** When LPDIV = 0 (divide-by-1), the internal bus clock and FB_CLK do not have a 50/50 duty cycle.

Table 9-6. CDR Field Description (continued)

Field	Description
7–6	Reserved, should be cleared.
5–0 SSIDIV	SSI baud clock divider. Specifies the divide value used to produce the baud clock for the SSI. This field is used only when MISCCR[SSISRC] is set (PLL is the source). $$\text{SSI Baud Clock} = \frac{f_{sys}}{\text{SSIDIV}/2} \qquad \textbf{\textit{Eqn. 9-2}}$$ **Note:** A value of 0 or 1 for SSIDIV represents a divide-by-62. SSIDIV should not be set to any value that sets the SSI baud clock frequency over the bus clock frequency (60 MHz or 80 MHz) because incorrect SSI operation could result.

9.3.6 USB Host Controller Status Register (UHCSR)

The UHCSR register controls and reflects various features of the USB host controller module. When the WKUP bit causes the assertion of an interrupt, that interrupt is cleared by a read of the UHCSR register.

Address: 0xFC0A_0014 (UHCSR)   Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PORTIND		0	0	0	0	0	0	0	0	0	0	0	WKUP	UHMIE	XPDE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
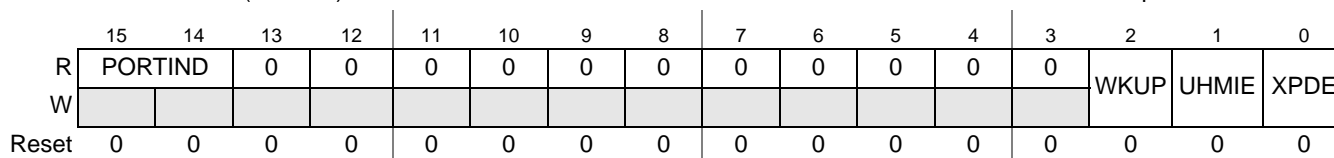
Figure 9-7. USB Host Controller Status Register (UHCSR)

Table 9-7. UHCSR Field Description

Field	Description
15–14 PORTIND	USB port indicator. Reflects the state of the host controller port indicator signals.
13–3	Reserved, should be cleared.
2 WKUP	USB host controller wakeup event. Reflects if a wakeup event has occurred on the USB host controller bus. When set, it generates an interrupt request if the UHMIE bit is set. 0 No outstanding wakeup event. 1 Wakup event has occurred.
1 UHMIE	USB host miscellaneous interrupt enable. Enables an interrupt to be generated from the WKUP bit. 0 WKUP interrupt disabled. 1 WKUP interrupt enabled.
0 XPDE	On-chip transceiver pull-down enable 0 50 kΩ pull-downs disabled on the host controller D+ and D- pins of the on-chip transceiver. 1 Optional on-chip 50 kΩ pull-downs enabled on the host controller D+ and D- pins of the on-chip transceiver.

9.3.7 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.

Address: 0xFC0A_0016 (UOCSR)   Access: Supervisor read/write
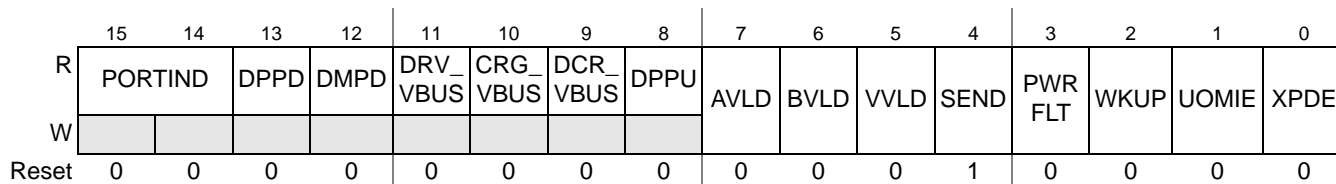
Figure 9-8. USB On-the-Go Controller Status Register (UOCSR)

Table 9-8. UOCSR Field Description

Field	Description
15–14 PORTIND	USB port indicator. Reflects the state of the OTG controller port indicator signals.
13 DPPD	D+ 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D+ line is active. When set, it 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, it asserts an interrupt if the UOMIE bit is set.
11 DRV_VBUS	Drive VBUS. Indicates the drive of the 5V power on VBUS is enabled.
10 CRG_VBUS	Charge VBUS. Indicates a charge resistor to pull-up VBUS is enabled. When set, it 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, it 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, it 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 that a power fault has occured on VBUS (e.g. overcurrent). 0 No power fault has occured. 1 Power fault has occured.
2 WKUP	USB OTG controller wakeup event. Reflects if a wakeup event has occured on the USB OTG Controller Bus 0 No outstanding wakeup event. 1 Wakup event has occurred.

Table 9-8. UOCSR Field Description (continued)

Field	Description
1 UOMIE	USB OTG miscellaneous interrupt enable. Enables an interrupt to be generated from any of the following bits of the UOCSR: DPPD, DMPD, DRV_VBUS, 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 Optional on-chip 50 kΩ pull-downs enabled on OTG D+ and D- transceiver pins of on-chip transceiver.

9.4   Functional Description

Six functions are defined within the chip configuration module:

1. Reset configuration
2. PLL mode
3. Oscillator mode
4. Boot device selection
5. Output pad strength configuration
6. Chip select configuration

These functions are described below.

9.4.1   Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states. Table 9-9 shows the states of the external pins while in reset.

Table 9-9. Reset Configuration Pin States During Reset

Pin	Pin Function[1]	I/O	Output State	Input State
D[15:0]	Primary function	Input	—	Must be driven by external logic
$\overline{\text{RCON}}$	$\overline{\text{RCON}}$ function for all modes[2]	Input	—	Internal weak pull-up device

[1]   If the external $\overline{\text{RCON}}$ pin is not asserted during reset, pin functions are determined by the default operation mode defined in the RCON register. If the external $\overline{\text{RCON}}$ pin is asserted, pin functions are determined by the override values driven on the external data bus pins.

[2]   During reset, the external $\overline{\text{RCON}}$ pin assumes its $\overline{\text{RCON}}$ pin function, but this pin changes to the function defined by the chip operation mode immediately after reset. See Table 9-10.

If the $\overline{\text{RCON}}$ pin is not asserted during reset, the chip configuration and the reset configuration pin functions after reset are determined by the RCON register or fixed defaults, 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.

If the $\overline{\text{RCON}}$ pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins. (See Table 9-10.) The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

NOTE

It is recommended that the logic levels for reset configuration on D[9:86:1] be actively driven when $\overline{\text{RCON}}$ is used. The rest of the data bus should be allowed to float or be pulled high.

Table 9-10. Configuration During Reset[1]

Pin(s) Affected	Default Configuration	Override Pins in Reset[2]	Function
None	RCON[1] = 0	**D1**	**PLL Mode**
		0	180/60 MHz operation (default)
		1	240/80 MHz operation
None	RCON[2] = 0	**D2**	**Oscillator Mode**
		0	Crystal oscillator mode (default)
		1	Oscillator bypass mode
None	RCON[4:3] = 00	**D[4:3]**	**Boot Device**
		00	External with 32-bit port[3] (default)
		01	External with 16-bit port
		10	External with 8-bit port
		11	External with 32-bit port
All output pins	RCON[5] = 0	**D5**	**Output pad drive strength**
		0	Low Drive Strength (default)
		1	High Drive Strength
None	RCON[6] = 0	**D6**	**Limp Mode**
		0	PLL mode (default)
		1	Limp mode
A[23:22]/$\overline{\text{FB\_CS}}$[5:4]	RCON[9:8] = 00	**D[9:8]**	**Chip Select Configuration**
		00	A[23:22] = A[23:22] (default)
		01	Reserved
		10	A23 = $\overline{\text{FB\_CS5}}$ and A22 = A22
		11	A[23:22] = $\overline{\text{FB\_CS}}$[5:4]

[1] Modifying the default configurations is possible only if the external $\overline{\text{RCON}}$ pin is asserted.

     [2] 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.

     [3] 32-bit port size is not available when DRAMSEL = 0. Defaults to 16-bit mode instead.

9.4.2 PLL Mode Selection

The initial device operating frequency is determined during reset configuration by the D1 pin. The default configuration with a 16 MHz input clock is 180 MHz for the core and 60 MHz for the internal bus. The user may choose to increase these frequencies (240 MHz and 80 MHz) by placing the device in limp mode and reconfiguring the appropriate PLL registers, or asserting D1 during reset configuration.

9.4.3 Oscillator Mode Selection

Use of the internal oscillator can be selected during reset configuration via the D2 pin. By default, the oscillator is enabled and the PLL is placed in normal mode with a crystal reference. If default configuration is over-ridden and D2 is asserted, the PLL is placed in normal mode with a external reference and the internal oscillator is bypassed. After reset is exited, the oscillator mode cannot be changed. See Chapter 7, "Clock Module" for more details on the available modes.

9.4.4 Boot Device Selection

During reset configuration, the $\overline{\text{FB\_CS0}}$ chip select pin is configured to select an external boot device. In this case, the V (valid) 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 that the reset vector loaded from address 0x0000_0004 causes the core to start executing from external memory space decoded by $\overline{\text{FB\_CS0}}$.

9.4.5 Output Pad Strength Configuration

Output pad strength is determined during reset configuration as shown in Table 9-11. After reset is exited, the output pad strength configuration can only be changed using the GPIO module. For more information see Chapter 13, "General Purpose I/O Module."

Table 9-11. Output Pad Driver Strength Selection[1]

Optional Pin Function Selection	D5
Output pads configured for low drive strength	D5 driven low
Output pads configured for full drive strength	D5 driven high

  [1] Modifying the default configurations is possible only if the external $\overline{\text{RCON}}$ pin is asserted low.

9.4.6 Chip Select Configuration

The chip select configuration ($\overline{FB\_CS}$[5:4]) is selected during reset and reflected in the CCR[CSC] field. After reset is exited, the chip select configuration cannot be changed. Table 9-10 shows the different chip select configurations that can be implemented during reset configuration.

Chapter 10
Reset Controller Module

10.1 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.

10.1.1 Block Diagram

Figure 10-1 illustrates the reset controller and is explained in the following sections.
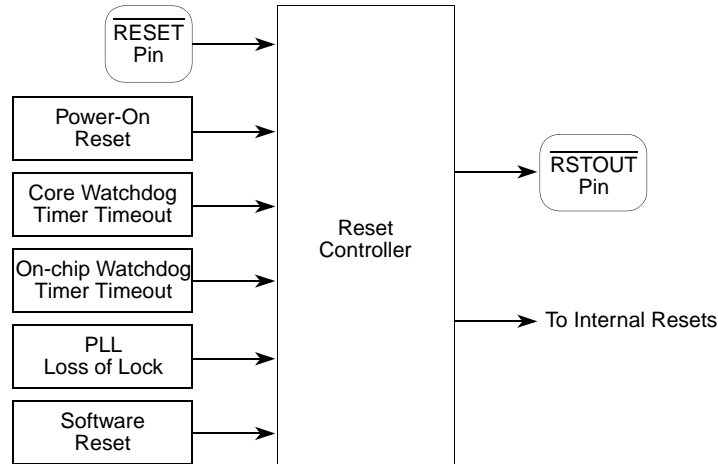
Figure 10-1. Reset Controller Block Diagram

10.1.2 Features

Module features include the following:

- Six sources of reset:
  — External
  — Power-on reset (POR)
  — Core watchdog timer
  — On-chip 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

10.2 External Signal Description

Table 10-1 provides a summary of the reset controller signal properties. The signals are described in the following paragraphs.

Table 10-1. Reset Controller Signal Properties

Name	Direction	Input Hysteresis	Input Synchronization
$\overline{\text{RESET}}$	I	Y	Y[1]
$\overline{\text{RSTOUT}}$	O	—	—

[1] $\overline{\text{RESET}}$ is always synchronized except when in low-power stop mode.

10.2.1 $\overline{\text{RESET}}$

Asserting the external $\overline{\text{RESET}}$ for at least four rising FB_CLK edges causes the external reset request to be recognized and latched.

10.2.2 $\overline{\text{RSTOUT}}$

This active-low output signal is driven low when the internal reset controller module resets the chip. 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 9, "Chip Configuration Module (CCM)," for more details on these override options.

10.3 Memory Map/Register Definition

The reset controller programming model consists of these registers:

- Reset control register (RCR), which selects reset controller functions
- Reset status register (RSR), which reflects the state of the last reset source

See Table 10-2 for the memory map and the following paragraphs for a description of the registers.

Table 10-2. Reset Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0000	Reset Control Register (RCR)	8	R/W	0x00	10.3.1/10-2
0xFC0A_0001	Reset Status Register (RSR)	8	R	See Section	10.3.2/10-3

10.3.1 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 10-2. Reset Control Register (RCR)

Table 10-3. 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.

10.3.2 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	WDR CHIP	POR	EXT	WDR CORE	LOL
W								
Reset:	0	0	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent

Figure 10-3. Reset Status Register (RSR)

Table 10-4. RSR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 SOFT	Software reset flag. Indicates that the last reset was caused by software. 0 Last reset not caused by software 1 Last reset caused by software

Field	Description
4 WDRCHIP	On-chip watchdog timer reset flag. Indicates that the last reset was caused by the on-chip watchdog timer timeout. 0 Last reset not caused by watchdog timer timeout 1 Last reset caused by watchdog timer timeout
3 POR	Power-on reset flag. Indicates that the last reset was caused by a power-on 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 that the last reset was caused by the core watchdog timer timeout. 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 that 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

10.4   Functional Description

10.4.1   Reset Sources

Table 10-5 defines the sources of reset and the signals driven by the reset controller.

Table 10-5. 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
On-chip watchdog timer	Synchronous
Loss of lock	Asynchronous
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 asserted immediately to the system.

10.4.1.1 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. Then after approximately another 512 cycles, $\overline{\text{RSTOUT}}$ is negated and the device begins operation.

10.4.1.2 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. The reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles after $\overline{\text{RESET}}$ is negated and the PLL has acquired lock. The device then exits reset and begins operation.

In low-power stop mode, the system clocks are stopped. Asserting the external $\overline{\text{RESET}}$ in stop mode causes an external reset to be recognized asynchronously.

10.4.1.3 On-chip Watchdog Timer Reset

An on-chip watchdog timer timeout causes 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 $\overline{\text{RSTOUT}}$ for approximately 512 cycles. Then the device exits reset and begins operation.

10.4.1.4 Core Watchdog Timer Reset

A core watchdog timer timeout causes 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 $\overline{\text{RSTOUT}}$ for approximately 512 cycles. Then the device exits reset and begins operation.

10.4.1.5 Loss-of-Lock Reset

This reset condition occurs when the device is operating in PLL mode and the PLL loses lock. The reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles after the PLL has acquired lock. The device then exits reset and resumes operation.

When operating in limp mode and the PLL loses lock (e.g. when changing PLL frequencies) this reset does not occur.

10.4.1.6 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 $\overline{\text{RSTOUT}}$ for approximately 512 cycles. Then the device exits reset and resumes operation.

10.4.2 Reset Control Flow

The reset logic control flow is shown in Figure 10-4. 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.
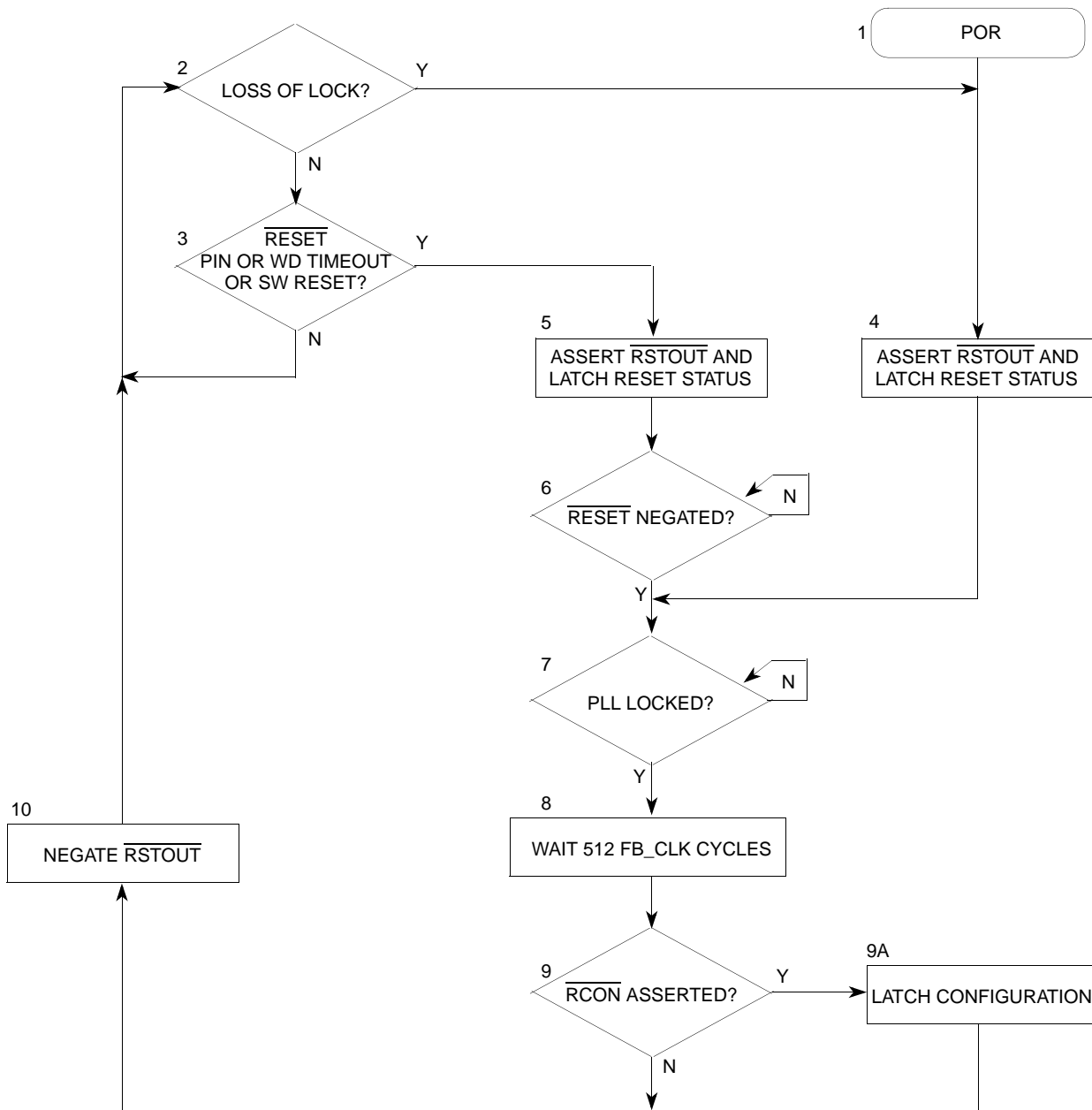
Figure 10-4. Reset Control Flow

10.4.2.1 Synchronous Reset Requests

In this discussion, the reference in parentheses refer to the state numbers in Figure 10-4. All cycle counts given are approximate.

If the external $\overline{\text{RESET}}$ signal is asserted by an external device 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 RSTOUT pin is asserted (5). 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 (8). The

reset control logic may latch the configuration according to the $\overline{\text{RCON}}$ signal level (9, 9A) before negating $\overline{\text{RSTOUT}}$ (10).

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 (8) or during the wait for PLL lock (7), the reset flow switches to (6) and waits for the $\overline{\text{RESET}}$ signal to be negated before continuing.

10.4.2.2 Internal Reset Request

If reset is asserted by an asynchronous internal reset source, such as loss of clock (1), the reset control logic asserts $\overline{\text{RSTOUT}}$ (4). The reset control logic waits for the PLL to attain lock (7) before waiting 512 FB_CLK cycles (8). Then the reset control logic may latch the configuration according to the $\overline{\text{RCON}}$ pin level (9, 9A) before negating $\overline{\text{RSTOUT}}$ (10).

If loss of lock occurs during the 512 count (8), the reset flow switches to (7) and waits for the PLL to lock before continuing.

10.4.2.3 Power-On Reset

When the reset sequence is initiated by power-on reset (1), the same reset sequence is followed as for the other asynchronous reset sources.

10.4.3 Concurrent Resets

This section describes the concurrent resets. As in the previous discussion references in parentheses refer to the state numbers in Figure 10-4.

10.4.3.1 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, 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, the reset sequence continues after a PLL lock (7).

10.4.3.2 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 sources of reset are asserted after the RSR status bits have been latched (4 or 5), the device is held in reset (8) until all sources have negated and the subsequent sources are not reflected in the RSR contents.

Chapter 11
System Control Module (SCM)

11.1 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.

11.1.1 Overview

The SCM provides programmable access protections for both 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.

NOTE
The core watchdog timer is available to provide compatibility with the watchdog timer implemented on previous ColdFire devices. However, there is a second watchdog timer available that has new features. See Chapter 27, "Watchdog Timer Module," for more information.

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.

11.1.2 Features

The SCM includes these distinctive features:

- Access control registers
    - Master privilege registers (MPR0 and MPR1)
    - 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

11.2 Memory Map/Register Definition

The memory map for the SCM registers is shown in Table 11-1.

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 only supports 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 11-1. SCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC00_0000[1]	Master Privilege Register 1 (MPR1)	32	R/W	0x7000_0007	11.2.1/11-3
0xEC00_0054[1]	Bus Monitor Timeout 1 (BMT1)	32	R/W	0x0000_0008	11.2.4/11-7
0xFC00_0000	Master Privilege Register 0 (MPR0)	32	R/W	0x7777_7777	11.2.1/11-3
0xFC00_0020	Peripheral Access Control Register A (PACRA)	32	R/W	0x5444_4444	11.2.3/11-4
0xFC00_0024	Peripheral Access Control Register B (PACRB)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0028	Peripheral Access Control Register C (PACRC)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_002C	Peripheral Access Control Register D (PACRD)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0040	Peripheral Access Control Register E (PACRE)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0044	Peripheral Access Control Register F (PACRF)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0048	Peripheral Access Control Register G (PACRG)	32	R/W	0x4444_4444	11.2.3/11-4
0xEC00_0040[1]	Peripheral Access Control Register H (PACRH)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0054	Bus Monitor Timeout 0 (BMT0)	32	R/W	0x0000_0008	11.2.4/11-7
0xFC04_0013	Wakeup Control Register (WCR)[2]	8	R/W	0x00	8.2.1/8-2
0xFC04_0016	Core Watchdog Control Register (CWCR)	16	R/W	0x0000	11.2.5/11-8
0xFC04_001B	Core Watchdog Service Register (CWSR)	8	R/W	Undefined	11.2.6/11-9
0xFC04_001F	SCM Interrupt Status Register (SCMISR)	8	R/W	0x00	11.2.7/11-10
0xFC04_0024	Burst Configuration Register (BCR)	32	R/W	0x0000_0000	11.2.8/11-10
0xFC04_0070	Core Fault Address Register (CFADR)	32	R	0x0000_0000	11.2.9/11-11
0xFC04_0075	Core Fault Interrupt Enable Register (CFIER)	8	R/W	0x00	11.2.10/11-12
0xFC04_0076	Core Fault Location Register (CFLOC)	8	R	Undefined	11.2.11/11-12
0xFC04_0077	Core Fault Attributes Register (CFATR)	8	R	Undefined	11.2.12/11-12
0xFC04_007C	Core Fault Data Register (CFDTR)	32	R	Undefined	11.2.13/11-13

[1] Take note of register location.

[2] The WCR register is described in Chapter 8, "Power Management."

11.2.1 Master Privilege Register 0 (MPR0)

The MPR specifies six 4-bit fields defining the access privilege level associated with a bus master in the device to the various peripherals listed in Table 11-4 (slots 0–48). The register provides one field per bus master.

Address: 0xFC00_0000 (MPR0)   Access: User read/write
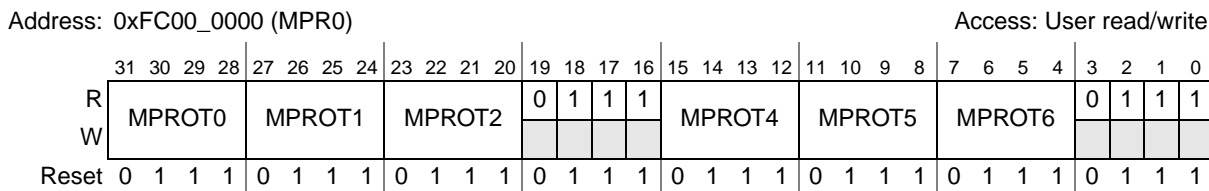
Figure 11-1. Master Privilege Register (MPR0)

Each master is assigned depending on its connection to the various cross-bar switch master ports.

Table 11-2. MPROT*n* Assignments

Cross-bar Switch Port Number	MPROT*n*	Master
M0	MPROT0	ColdFire Core
M1	MPROT1	eDMA Controller
M2	MPROT2	FEC
M4	MPROT4	LCD Controller
M5	MPROT5	USB Host
M6	MPROT6	USB On-the-Go

The MPROT*n* field is defined as shown below.
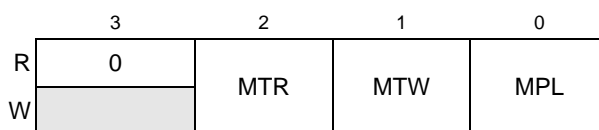
Figure 11-2. MPROT*n* Fields

Table 11-3. MPROT*n* Field Descriptions

Field	Description
3	Reserved, should 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.

Table 11-3. MPROT*n* Field Descriptions (continued)

Field	Description
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.

11.2.2 Master Privilege Register 1 (MPR1)

Similar to the MPR0 register, the MPR1 register specifies six fields defining the access privilege level associated with a bus master in the device. However, this register specifies master access levels for only the cryptography modules, MDHA, SKHA, and RNG (slots 56–58). For register bit descriptions see Section 11.2.1, "Master Privilege Register 0 (MPR0)."

Address: 0xEC00_0000 (MPR1)   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				0	0	0	0	MPROT4				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 11-3. Master Privilege Register 1 (MPR1)

11.2.3 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	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
W																																
Reset	0	1	0	1	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 11-4. 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	PACR8				0	1	0	0	0	1	0	0	0	1	0	0	PACR12				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 11-5. 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	1	0	0	PACR21	PACR22	PACR23
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 11-6. 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 1 0 0	PACR28	PACR29	PACR30	PACR31
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 11-7. 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	PACR36	PACR37	PACR38	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 11-8. 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	PACR40	PACR41	PACR42	PACR43	PACR44	PACR45	PACR46	PACR47
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 11-9. 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	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
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 11-10. Peripheral Access Control Register G (PACRG)

Address: 0xEC00_0040 (PACRH) 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	PACR56	PACR57	PACR58	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 11-11. Peripheral Access Control Register H (PACRH)

Each peripheral is assigned to its PACR*n* field as shown below:

MCF5329 Reference Manual, Rev 3

Table 11-4. PACR*n* Assignments

Slot Number	PACR*n*	Peripheral
0	PACR0	SCM (MPR & PACRs)
1	PACR1	Cross-bar switch
2	PACR2	FlexBus
8	PACR8	FlexCAN
12	PACR12	FEC
16	PACR16	SCM (CWT & 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	QSPI
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
32	PACR32	PIT 0
33	PACR33	PIT 1
34	PACR34	PIT 2
35	PACR35	PIT 3
36	PACR36	PWM
37	PACR37	Edge Port
38	PACR38	On-chip Watchdog Timer
40	PACR40	CCM, Reset Controller, Power Management
41	PACR41	GPIO Module
42	PACR42	Real Time Clock
43	PACR43	LCD Controller
44	PACR44	USB On-the-Go
45	PACR45	USB Host
46	PACR46	SDRAM Controller

Table 11-4. PACR*n* Assignments (continued)

Slot Number	PACR*n*	Peripheral
47	PACR47	SSI
48	PACR48	PLL
56	PACR56	MDHA
57	PACR57	SKHA
58	PACR58	RNG

The PACR*n* field is defined as shown below.
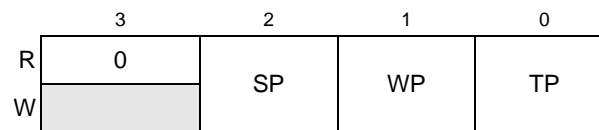
Figure 11-12. PACR*n* Fields

Table 11-5. PACR*n* Field Descriptions

Field	Description
3	Reserved, should 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, the access is terminated with an error response and no peripheral access is initiated.
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, the access is terminated with an error response and no peripheral access is initiated.
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 is terminated with an error response and no peripheral access is initiated.

11.2.4 Bus Monitor Timeout Registers (BMT*n*)

Each AIPS controller implements a bus timeout monitor to insure that every bus cycle is properly terminated within a programmed period of time. The monitor continually checks for termination of each bus cycle and completes the cycle if there is no response when the programmed monitor cycle count is reached.

The monitor can be programmed from 8–1024 internal bus cycles under control of the BMT register. If the programmed timeout value is reached before a termination, the bus monitor completes the cycle with an error termination. Thus, the SCMISR[CFEI] bit is set, and an interrupt to the interrupt controller is generated if the CFIER[ECFEI] bit is set. At reset, the BMT is enabled with a maximum timeout value.

Address: 0xEC00_0054 (BMT1)   Access: User read/write
         0xFC00_0054 (BMT0)

	31	30	29	28	27	26	25	24	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	BME	BMT		
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
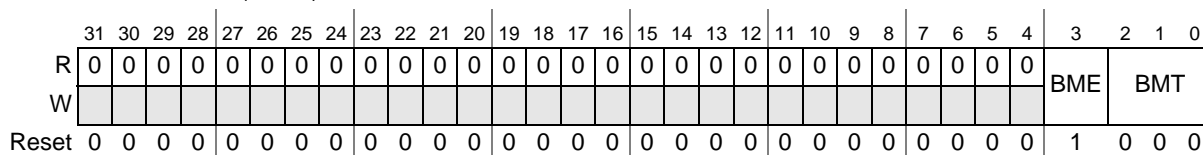
Figure 11-13. Bus Monitor Timeout Registers (BMT*n*)

Table 11-6. BMT Field Descriptions

Field	Description
31–4	Reserved, should be cleared.
3 BME	Bus monitor timeout enable. 0 BMT disabled 1 BMT enabled
2–0 BMT	Bus monitor timeout period. Indicates the timeout period in internal bus cycles for the bus monitor. 000 1024 cycles 001 512 cycles 010 256 cycles 011 128 cycles 100 64 cycles 101 32 cycles 110 16 cycles 111 8 cycles

11.2.5   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
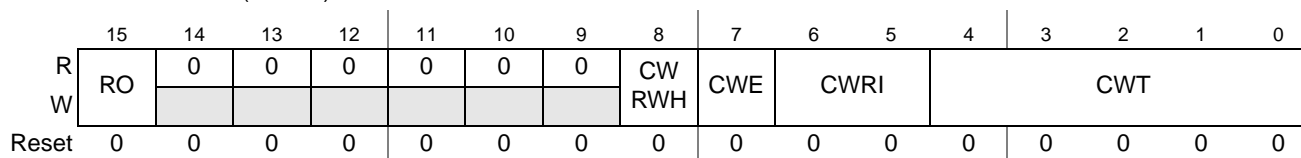
Figure 11-14. Core Watchdog Control Register (CWCR)

Table 11-7. 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, should be cleared.

Table 11-7. CWCR Field Descriptions (continued)

Field	Description
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.
6–5 CWRI	Core watchdog reset/interrupt. 00 If a time-out occurs, the CWT generates an interrupt to the core. Refer to Chapter 14, "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 theRSR[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 since CWCR[CWE] = 0 at reset. For CWCR[CWT] = n, time-out period = 2^n system clock cycles, where n = 8–31. If $n < 8$, then time-out period is forced to 2^8.

11.2.6 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: first 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 11-15 illustrates the CWSR. At system reset, the contents of CWSR are uninitialized.

NOTE

If the CWT is enabled and has not timed out, then 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					CWSR			
W								
Reset:	—	—	—	—	—	—	—	—
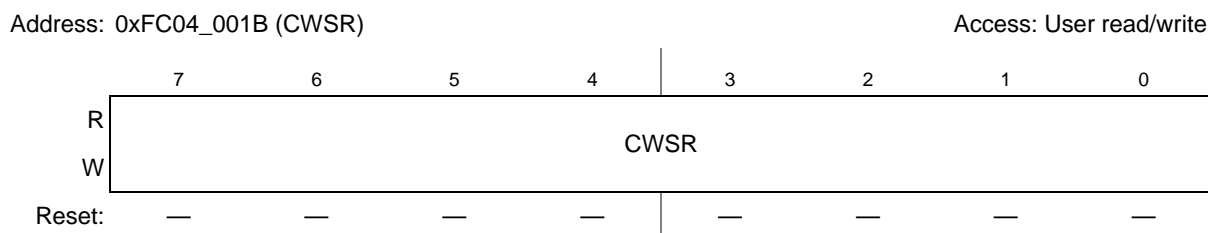
Figure 11-15. Core Watchdog Service Register (CWSR)

11.2.7 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 will only be sent 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
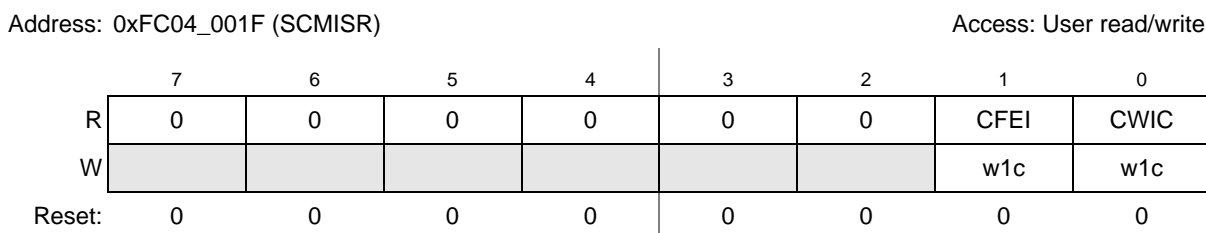
Figure 11-16. SCM Interrupt Status Register (SCMISR)

Table 11-8. SCMISR Field Descriptions

Field	Description
7–2	Reserved, should be cleared.
1 CFEI	Core fault error interrupt flag. Indicates if a bus fault has occurred. 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 only enabled if CFLOC[ECFEI] is set. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect. **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. 0　No CWT interrupt has occurred. 1　CWT interrupt has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect.

11.2.8 Burst Configuration Register (BCR)

The BCR register is used to enable or disable the LCD controller, USB host, and USB On-the-Go modules for bursting to/from the cross-bar 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
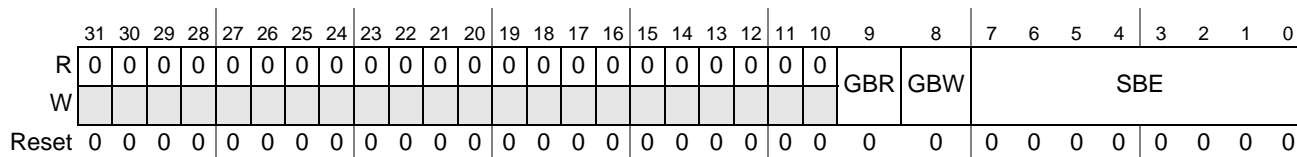
Figure 11-17. Burst Configuration Register (BCR)

Table 11-9. BCR Field Descriptions

Field	Description
31–10	Reserved, should be cleared.
9 GBR	Global burst enable for reads. Allows bursts to happen on read transactions from the crossbar switch slaves to the LCD controller, USB host, and USB On-the-Go modules. 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 LCD controller, USB host, and USB On-the-Go modules. 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.

11.2.9 Core Fault Address Register (CFADR)

The CFADR is a read-only register for indicating the address of the last core access which was 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
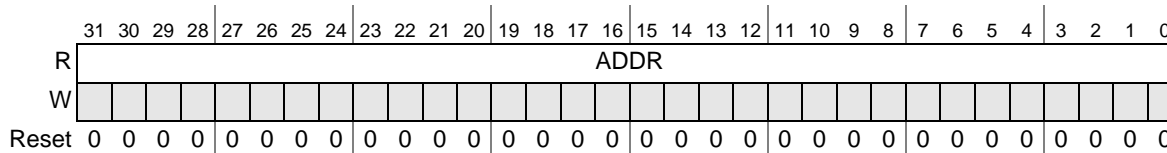
Figure 11-18. Core Fault Address Register (CFADR)

Table 11-10. CFADR Field Descriptions

Field	Description
31–0 ADDR	Indicates the faulting address of the last core access terminated with an error response.

11.2.10 Core Fault Interrupt Enable Register (CFIER)

The CFIER register is used to enable the system bus error interrupt. See Chapter 14, "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
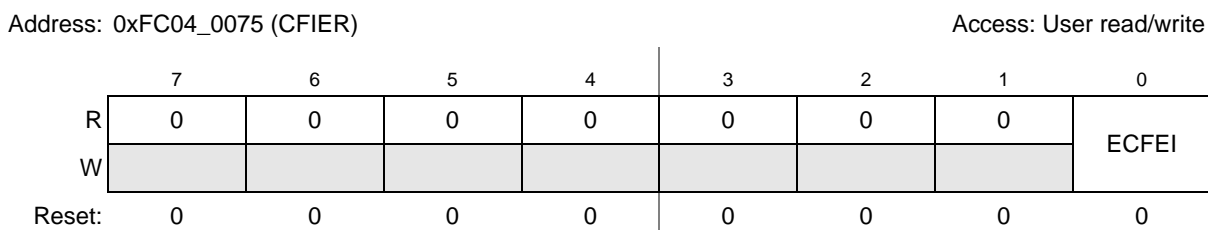
Figure 11-19. Core Fault Interrupt Enable Register (CFIER)

Table 11-11. CFIER Field Descriptions

Field	Description
7–1	Reserved, should 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.

11.2.11 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
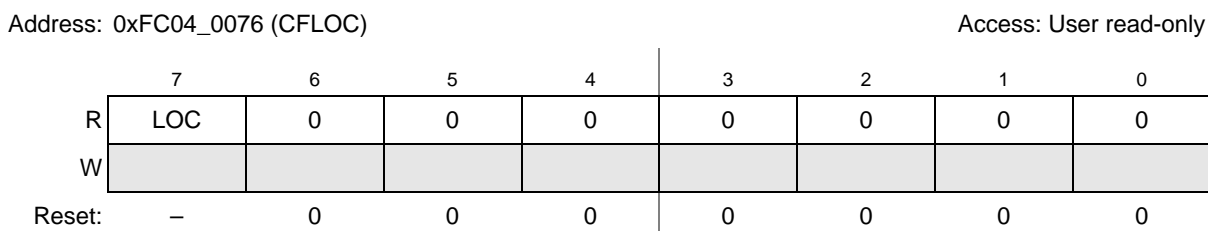
Figure 11-20. Core Fault Location Register (CFLOC)

Table 11-12. 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, should be cleared.

11.2.12 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:	–	–	–	–	–	–	–	–
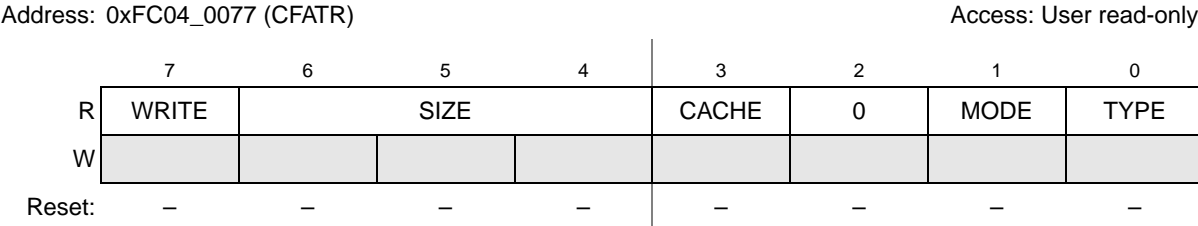
Figure 11-21. Core Fault Attributes Register (CFATR)

Table 11-13. 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, should 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

11.2.13 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 only valid for faulted internal bus write accesses, CFLOC[LOC] = 0.

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	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
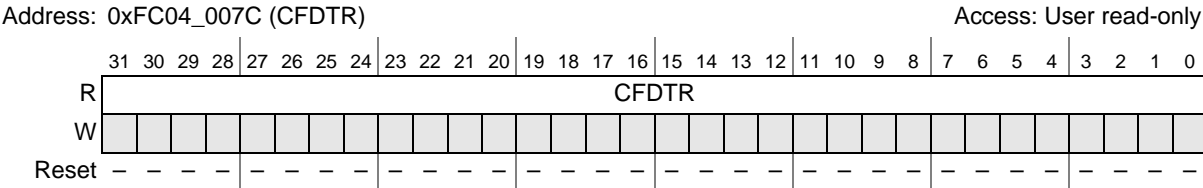
Figure 11-22. Core Fault Data Register (CFDTR)

Table 11-14. CFDTR Field Descriptions

Field	Description
31–0 CFDTR	Contains the data associated with the faulting access of the last internal bus write access. The register contains the data value taken directly from the write data bus.

11.3   Functional Description

11.3.1   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.

11.3.2   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:

1. An interrupt may be generated to the core.
2. An immediate system reset.
3. 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.
4. 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 be executed 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, then any write of a data value other than 0x55 or 0xAA will cause 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, there is a write-once control bit in the CWCR that sets the CWCR to read-only to prevent accidental updates to this control register from changing the desired system configuration. Once 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.

11.3.3   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 that is terminated with an error.

Chapter 12
Crossbar Switch (XBS)

12.1 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 MCF5329 devices have up to seven masters and four slaves (7Mx4S) connected to the crossbar switch. The seven masters are the ColdFire core, eDMA controller, FEC, LCD controller, USB host, USB OTG modules, and a reserved master for factory test. The slaves are FlexBus/SDRAM controller, SRAM controller, cryptography modules,and the peripheral bus controller.

Figure 12-1 is a block diagram of the MCF5329 family bus architecture showing the crossbar switch configuration.
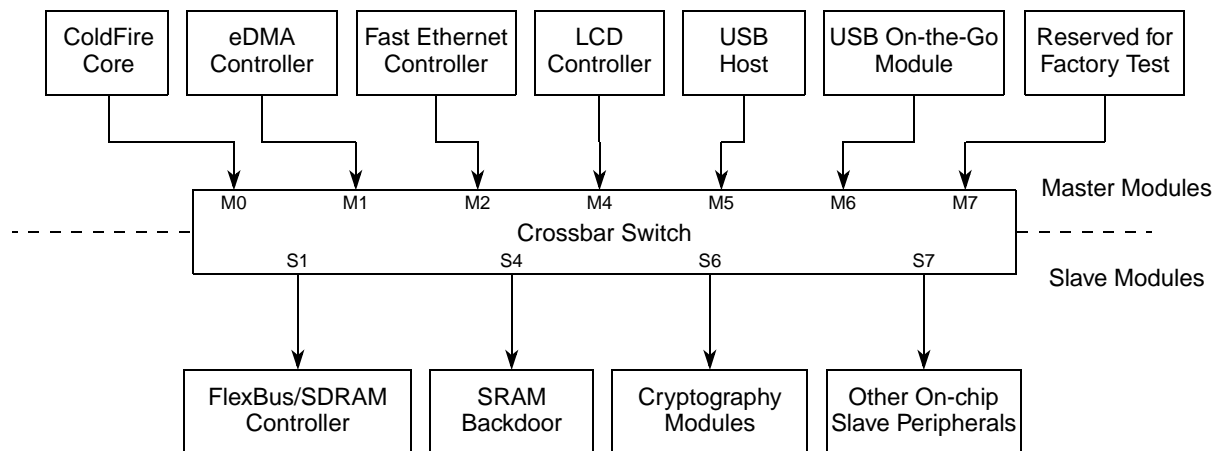
Figure 12-1. Bus Architecture Block Diagram

The modules are assigned to the numbered ports on the crossbar switch in the below table.

Table 12-1. 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	
Master 4 (M4)	LCD controller	
Master 5 (M5)	USB host	
Master 6 (M6)	USB On-the-Go	
Master 7 (M7)	Reserved for factory test	
Slave Modules		
Crossbar Port	**Module**	**Address Range[1]**
Slave 1 (S1)	Flexbus SDRAM Controller	0x0000_0000–0x3FFF_FFFF & 0xC000_0000–0xDFFF_FFFF 0x4000_0000–0x7FFF_FFFF
Slave 4 (S4)	Internal SRAM Backdoor	0x8000_0000–0x8FFF_FFFF
Slave 6 (S6)	Cryptography Modules (RNG, SKHA, MDHA)	0xE000_0000–0xEFFF_FFFF[2]
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.

12.2 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

12.3 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 12.5.1.1, "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 12.5.1.2, "Round-Robin Priority Operation."

12.4 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 11.2.7, "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 12-2 shows the memory map for the crossbar switch program-visible registers.

Table 12-2. XBS Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4100	Priority Register Slave 1 (XBS_PRS1)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4110	Control Register Slave 1 (XBS_CRS1)	32	R/W	0x0000_0000	12.4.2/12-5
0xFC00_4400	Priority Register Slave 4 (XBS_PRS4)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4410	Control Register Slave 4 (XBS_CRS4)	32	R/W	0x0000_0000	12.4.2/12-5

Table 12-2. XBS Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4600	Priority Register Slave 6 (XBS_PRS6)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4610	Control Register Slave 6 (XBS_CRS6)	32	R/W	0x0000_0000	12.4.2/12-5
0xFC00_4700	Priority Register Slave 7 (XBS_PRS7)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4710	Control Register Slave 7 (XBS_CRS7)	32	R/W	0x0000_0000	12.4.2/12-5

12.4.1 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 11.2.7, "SCM Interrupt Status Register (SCMISR)") and the XBS_PRS*n* is not updated.

Address: 0xFC00_4100 (XBS_PRS1) Access: Supervisor read/write
0xFC00_4400 (XBS_PRS4)
0xFC00_4600 (XBS_PRS6)
0xFC00_4700 (XBS_PRS7)

	31	30	29	28	27	26	25	24	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		M4		0	0	0	0	0		M2		0		M1		0		M0	
W																																
Reset	0	1	1	0	0	1	0	1	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

Figure 12-2. XBS Priority Registers Slave *n* (XBS_PRS*n*)

Table 12-3. XBS_PRS*n* Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–28 M7	Master 7 (Factory Test) 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.

Table 12-3. XBS_PRS*n* Field Descriptions (continued)

Field	Description
23	Reserved, must be cleared.
22–20 M5	Master 5 (USB Host) priority. See M7 description.
19	Reserved, must be cleared.
18–16 M4	Master 4 (LCD Controller) priority. See M7 description.
15–11	Reserved, must be cleared.
10–8 M2	Master 2 (FEC) 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 (including reserved masters), valid values are 000 to 110. Unpredictable results occur when using the reserved setting of 111.

12.4.2 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) Access: Supervisor read/write
0xFC00_4410 (XBS_PRS4)
0xFC00_4610 (XBS_PRS6)
0xFC00_4710 (XBS_PRS7)
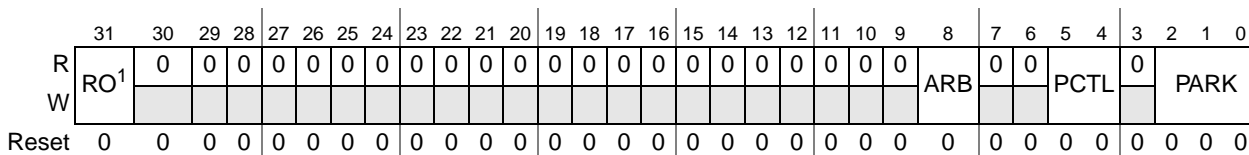
[1] After this bit is set, only a hardware reset clears it.

Figure 12-3. XBS Control Registers Slave *n* (XBS_CRS*n*)

Table 12-4. 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, onlya hardware reset clears it. 0 Both of the slave port's registers are writteable. 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 (FEC) 011 Reserved 100 Park on master port M4 (LCD Controller) 101 Park on master port M5 (USB Host) 110 Park on master port M6 (USB OTG) 111 Reserved

12.5   Functional Description

12.5.1   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.

12.5.1.1   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.

12.5.1.2 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.

12.5.1.3 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 11.2.7, "SCM Interrupt Status Register (SCMISR)") and the registers are not updated.

12.6 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 13
General Purpose I/O Module

13.1 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 13-1 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 below figure only for completeness.
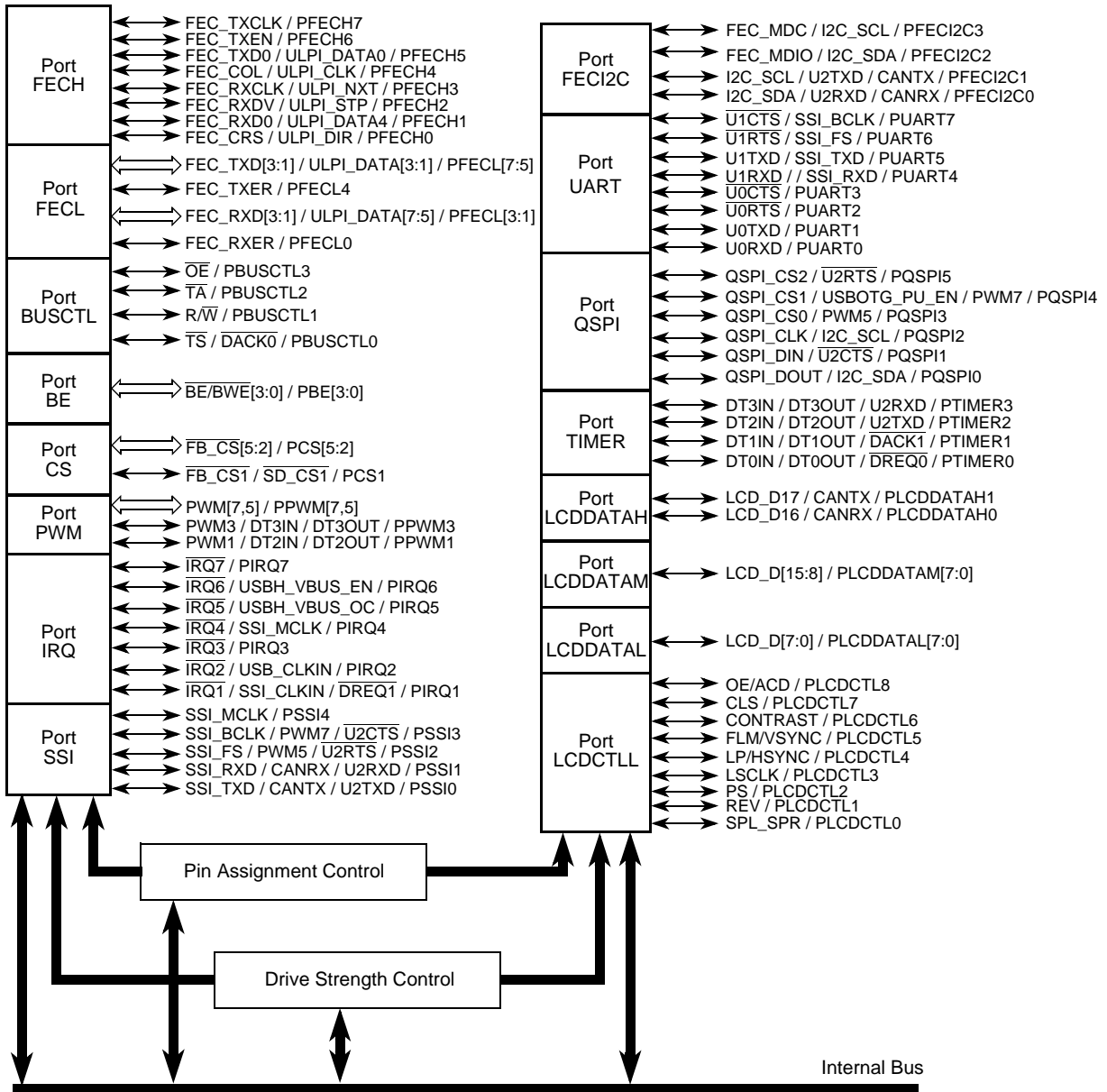
Figure 13-1. Ports Module Block Diagram

13.1.1 Overview

The GPIO module controls the configuration for various external pins, including those used for:

- External bus accesses
- External chip selection
- Ethernet data and control
- I^2C serial control
- QSPI
- 32-bit DMA timers

MCF5329 Reference Manual, Rev 3

- FEC
- UART
- SSI
- LCD Controller

13.1.2   Features

The ports module includes these distinctive features:

- Control of primary function use
  - On all supported GPIO ports
  - On pins whose GPIO is not supported by ports module: $\overline{IRQ}$[6:4], $\overline{IRQ}$[2:1]
- 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

13.2   External Signal Description

The GPIO module controls the functionality of several external pins. These pins are listed in Table 13-2 under the GPIO column.

After reset ports BUSCTL, BE and CS are configured for external memory. They are available for the user as GPIO if the corresponding registers are set appropriately. All other ports default to GPIO after reset.

NOTE
In this table and throughout this document, a single signal within a group is designated without square brackets (i.e., A23), while designations for multiple signals within a group use brackets (i.e., A[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. Pins that are muxed with GPIO default to their GPIO functionality.

Table 13-1. MCF5327/8/9 Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
Reset								
$\overline{RESET}$[2]	—	—	—	I	EVDD	J11	N15	N15

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
$\overline{\text{RSTOUT}}$	—	—	—	O	EVDD	P14	P14	P14
Clock								
EXTAL	—	—	—	I	EVDD	L14	P16	P16
XTAL[2]	—	—	—	O	EVDD	K14	N16	N16
EXTAL32K	—	—	—	I	EVDD	M11	P13	P13
XTAL32K	—	—	—	O	EVDD	N11	R13	R13
FB_CLK	—	—	—	O	SDVDD	L1	T2	T2
Mode Selection								
$\overline{\text{RCON}}$[2]	—	—	—	I	EVDD	M7	M8	M8
DRAMSEL	—	—	—	I	EVDD	G11	H12	H12
FlexBus								
A[23:22]	—	$\overline{\text{FB\_CS}}$[5:4]	—	O	SDVDD	B11,C11	C13, D13	C13, D13
A[21:16]	—	—	—	O	SDVDD	B12, A12, D11, C12, B13, A13	E13, A14, B14, C14, A15, B15	E13, A14, B14, C14, A15, B15
A[15:14]	—	SD_BA[1:0][3]	—	O	SDVDD	A14, B14	D14, B16	D14, B16
A[13:11]	—	SD_A[13:11][3]	—	O	SDVDD	C13, C14, D12	C15, C16, D15	C15, C16, D15
A10	—	—	—	O	SDVDD	D13	D16	D16
A[9:0]	—	SD_A[9:0][3]	—	O	SDVDD	D14, E11–14, F11–F14, G14	E14–E16, F13–F16, G16– G14	E14–E16, F13–F16, G16– G14
D[31:16]	—	SD_D[31:16][4]	—	I/O	SDVDD	H3–H1, J4–J1, K1, L4, M2, M3, N1, N2, P1, P2, N3	M1–M4, N1–N4, T3, P4, R4, T4, N5, P5, R5, T5	M1–M4, N1–N4, T3, P4, R4, T4, N5, P5, R5, T5
D[15:1]	—	FB_D[31:17][4]	—	I/O	SDVDD	F4–F1, G5–G2, L5, N4, P4, M5, N5, P5, L6	J3–J1, K4–K1, L2, R6, N7, P7, R7, T7, P8, R8	J3–J1, K4–K1, L2, R6, N7, P7, R7, T7, P8, R8
D0[2]	—	FB_D[16][4]	—	I/O	SDVDD	M6	T8	T8
$\overline{\text{BE/BWE}}$[3:0]	PBE[3:0]	$\overline{\text{SD\_DQM}}$[3:0][3]	—	O	SDVDD	H4, P3, G1, M4	L4, P6, L3, N6	L4, P6, L3, N6

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
$\overline{OE}$	PBUSCTL3	—	—	O	SDVDD	P6	R9	R9
$\overline{TA}$[2]	PBUSCTL2	—	—	I	SDVDD	G13	G13	G13
R/$\overline{W}$	PBUSCTL1	—	—	O	SDVDD	N6	N8	N8
$\overline{TS}$	PBUSCTL0	$\overline{DACK0}$	—	O	SDVDD	D2	H4	H4
Chip Selects								
$\overline{FB\_CS}$[5:4]	PCS[5:4]	—	—	O	SDVDD	—	B13, A13	B13, A13
$\overline{FB\_CS}$[3:1]	PCS[3:1]			O	SDVDD	A11, D10, C10	A12, B12, C12	A12, B12, C12
$\overline{FB\_CS0}$	—	—	—	O	SDVDD	B10	D12	D12
SDRAM Controller								
SD_A10	—	—	—	O	SDVDD	L2	P2	P2
SD_CKE	—	—	—	O	SDVDD	E1	H2	H2
SD_CLK	—	—	—	O	SDVDD	K3	R1	R1
$\overline{SD\_CLK}$	—	—	—	O	SDVDD	K2	R2	R2
$\overline{SD\_CS1}$	—	—	—	O	SDVDD	—	J4	J4
$\overline{SD\_CS0}$	—	—	—	O	SDVDD	E2	H1	H1
SD_DQS3	—	—	—	O	SDVDD	H5	L1	L1
SD_DQS2	—	—	—	O	SDVDD	K6	T6	T6
$\overline{SD\_SCAS}$	—	—	—	O	SDVDD	L3	P3	P3
$\overline{SD\_SRAS}$	—	—	—	O	SDVDD	M1	R3	R3
SD_SDR_DQS	—	—	—	O	SDVDD	K4	P1	P1
$\overline{SD\_WE}$	—	—	—	O	SDVDD	D1	H3	H3
External Interrupts Port[5]								
$\overline{IRQ7}$[2]	PIRQ7[2]	—	—	I	EVDD	J13	J13	J13
$\overline{IRQ6}$[2]	PIRQ6[2]	USBHOST_ VBUS_EN	—	I	EVDD	—	J14	J14
$\overline{IRQ5}$[2]	PIRQ5[2]	USBHOST_ VBUS_OC	—	I	EVDD	—	J15	J15
$\overline{IRQ4}$[2]	PIRQ4[2]	SSI_MCLK	—	I	EVDD	L13	J16	J16
$\overline{IRQ3}$[2]	PIRQ3[2]	—	—	I	EVDD	M14	K14	K14
$\overline{IRQ2}$[2]	PIRQ2[2]	USB_CLKIN	—	I	EVDD	M13	K15	K15

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
$\overline{IRQ1}$[2]	PIRQ1[2]	$\overline{DREQ1}$[2]	SSI_CLKIN	I	EVDD	N13	K16	K16
FEC								
FEC_MDC	PFECI2C3	I2C_SCL[2]	—	O	EVDD	—	C1	C1
FEC_MDIO	PFECI2C2	I2C_SDA[2]	—	I/O	EVDD	—	C2	C2
FEC_TXCLK	PFECH7	—	—	I	EVDD	—	A2	A2
FEC_TXEN	PFECH6	—	—	O	EVDD	—	B2	B2
FEC_TXD0	PFECH5	ULPI_DATA0	—	O	EVDD	—	E4	E4
FEC_COL	PFECH4	ULPI_CLK	—	I	EVDD	—	A8	A8
FEC_RXCLK	PFECH3	ULPI_NXT	—	I	EVDD	—	C8	C8
FEC_RXDV	PFECH2	ULPI_STP	—	I	EVDD	—	D8	D8
FEC_RXD0	PFECH1	ULPI_DATA4	—	I	EVDD	—	C6	C6
FEC_CRS	PFECH0	ULPI_DIR	—	I	EVDD	—	B8	B8
FEC_TXD[3:1]	PFECL[7:5]	ULPI_DATA[3:1]	—	O	EVDD	—	D3–D1	D3–D1
FEC_TXER	PFECL4	—	—	O	EVDD	—	B1	B1
FEC_RXD[3:1]	PFECL[3:1]	ULPI_DATA[7:5]	—	I	EVDD	—	E7, A6, B6	E7, A6, B6
FEC_RXER	PFECL0	—	—	I	EVDD	—	D4	D4
LCD Controller								
LCD_D17	PLCDDH1	CANTX	—	O	EVDD	—	—	C9
LCD_D16	PLCDDH0	CANRX	—	O	EVDD	—	—	D9
LCD_D17	PLCDDH1	—	—	O	EVDD	A6	C9	—
LCD_D16	PLCDDH0	—	—	O	EVDD	B6	D9	—
LCD_D15	PLCDDM7	—	—	O	EVDD	C6	A7	A7
LCD_D14	PLCDDM6	—	—	O	EVDD	D6	B7	B7
LCD_D13	PLCDDM5	—	—	O	EVDD	A5	C7	C7
LCD_D12	PLCDDM4	—	—	O	EVDD	B5	D7	D7
LCD_D[11:8]	PLCDDM[3:0]	—	—	O	EVDD	C5, D5, A4, B4	D6, E6, A5, B5	D6, E6, A5, B5
LCD_D7	PLCDDL7	—	—	O	EVDD	C4	C5	C5
LCD_D6	PLCDDL6	—	—	O	EVDD	B3	D5	D5
LCD_D5	PLCDDL5	—	—	O	EVDD	A3	A4	A4
LCD_D4	PLCDDL4	—	—	O	EVDD	A2	A3	A3

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
LCD_D[3:0]	PLCDDL[3:0]	—	—	O	EVDD	D4, C3, D3, B2	B4, C4, B3, C3	B4, C4, B3, C3
LCD_ACD/ LCD_OE	PLCDCTLH0	—	—	O	EVDD	D7	B9	B9
LCD_CLS	PLCDCTLL7	—	—	O	EVDD	C7	A9	A9
LCD_CONTRAST	PLCDCTLL6	—	—	O	EVDD	B7	D10	D10
LCD_FLM/ LCD_VSYNC	PLCDCTLL5	—	—	O	EVDD	A7	C10	C10
LCD_LP/ LCD_HSYNC	PLCDCTLL4	—	—	O	EVDD	A8	B10	B10
LCD_LSCLK	PLCDCTLL3	—	—	O	EVDD	B8	A10	A10
LCD_PS	PLCDCTLL2	—	—	O	EVDD	C8	A11	A11
LCD_REV	PLCDCTLL1	—	—	O	EVDD	D8	B11	B11
LCD_SPL_SPR	PLCDCTLL0	—	—	O	EVDD	B9	C11	C11
USB Host & USB On-the-Go								
USBOTG_M	—	—	—	I/O	USB VDD	G12	L15	L15
USBOTG_P	—	—	—	I/O	USB VDD	H13	L16	L16
USBHOST_M	—	—	—	I/O	USB VDD	K13	M15	M15
USBHOST_P	—	—	—	I/O	USB VDD	J12	M16	M16
FlexCAN (MCF53281 & MCF5329 only)								
CANRX and CANTX do not have dedicated bond pads. Please refer to the following pins for muxing: I2C_SDA, SSI_RXD, or LCD_D16 for CANRX and I2C_SCL, SSI_TXD, or LCD_D17 for CANTX.								
PWM								
PWM7	PPWM7	—	—	I/O	EVDD	—	H13	H13
PWM5	PPWM5	—	—	I/O	EVDD	—	H14	H14
PWM3	PPWM3	DT3OUT	DT3IN	I/O	EVDD	H14	H15	H15
PWM1	PPWM1	DT2OUT	DT2IN	I/O	EVDD	J14	H16	H16
SSI								
SSI_MCLK	PSSI4	—	—	I/O	EVDD	—	G4	G4
SSI_BCLK	PSSI3	$\overline{U2CTS}$	PWM7	I/O	EVDD	—	F4	F4

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
SSI_FS	PSSI2	$\overline{\text{U2RTS}}$	PWM5	I/O	EVDD	—	G3	G3
SSI_RXD[2]	PSSI1	U2RXD	CANRX	I	EVDD	—	—	G2
SSI_TXD[2]	PSSI0	U2TXD	CANTX	O	EVDD	—	—	G1
SSI_RXD[2]	PSSI1	U2RXD	—	I	EVDD	—	G2	—
SSI_TXD[2]	PSSI0	U2TXD	—	O	EVDD	—	G1	—
I²C								
I2C_SCL[2]	PFECI2C1	CANTX	U2TXD	I/O	EVDD	—	—	F3
I2C_SDA[2]	PFECI2C0	CANRX	U2RXD	I/O	EVDD	—	—	F2
I2C_SCL[2]	PFECI2C1	—	U2TXD	I/O	EVDD	E3	F3	—
I2C_SDA[2]	PFECI2C0	—	U2RXD	I/O	EVDD	E4	F2	—
DMA								
$\overline{\text{DACK}}$[1:0] and $\overline{\text{DREQ}}$[1:0] do not have dedicated bond pads. Please refer to the following pins for muxing: $\overline{\text{TS}}$ for $\overline{\text{DACK0}}$, DT0IN for $\overline{\text{DREQ0}}$, DT1IN for $\overline{\text{DACK1}}$, and $\overline{\text{IRQ1}}$ for $\overline{\text{DREQ1}}$.								
QSPI								
QSPI_CS2	PQSPI5	$\overline{\text{U2RTS}}$	—	O	EVDD	P10	T12	T12
QSPI_CS1	PQSPI4	PWM7	USBOTG_PU_EN	O	EVDD	L11	T13	T13
QSPI_CS0	PQSPI3	PWM5	—	O	EVDD	—	P11	P11
QSPI_CLK	PQSPI2	I2C_SCL[2]	—	O	EVDD	N10	R12	R12
QSPI_DIN	PQSPI1	$\overline{\text{U2CTS}}$	—	I	EVDD	L10	N12	N12
QSPI_DOUT	PQSPI0	I2C_SDA	—	O	EVDD	M10	P12	P12
UARTs								
$\overline{\text{U1CTS}}$	PUARTL7	SSI_BCLK	—	I	EVDD	C9	D11	D11
$\overline{\text{U1RTS}}$	PUARTL6	SSI_FS	—	O	EVDD	D9	E10	E10
U1TXD	PUARTL5	SSI_TXD[2]	—	O	EVDD	A9	E11	E11
U1RXD	PUARTL4	SSI_RXD[2]	—	I	EVDD	A10	E12	E12
$\overline{\text{U0CTS}}$	PUARTL3	—	—	I	EVDD	P13	R15	R15
$\overline{\text{U0RTS}}$	PUARTL2	—	—	O	EVDD	N12	T15	T15
U0TXD	PUARTL1	—	—	O	EVDD	P12	T14	T14
U0RXD	PUARTL0	—	—	I	EVDD	P11	R14	R14

Note: The UART2 signals are multiplexed on the QSPI, SSI, DMA Timers, and I2C pins.

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
DMA Timers								
DT3IN	PTIMER3	DT3OUT	U2RXD	I	EVDD	C1	F1	F1
DT2IN	PTIMER2	DT2OUT	U2TXD	I	EVDD	B1	E1	E1
DT1IN	PTIMER1	DT1OUT	$\overline{DACK1}$	I	EVDD	A1	E2	E2
DT0IN	PTIMER0	DT0OUT	$\overline{DREQ0}$[2]	I	EVDD	C2	E3	E3
BDM/JTAG[6]								
JTAG_EN[7]	—	—	—	I	EVDD	L12	M13	M13
DSCLK	—	$\overline{TRST}$[2]	—	I	EVDD	N14	P15	P15
PSTCLK	—	TCLK[2]	—	O	EVDD	L7	T9	T9
$\overline{BKPT}$	—	TMS[2]	—	I	EVDD	M12	R16	R16
DSI	—	TDI[2]	—	I	EVDD	K12	N14	N14
DSO	—	TDO	—	O	EVDD	N9	N11	N11
DDATA[3:0]	—	—	—	O	EVDD	N7, P7, L8, M8	N9, P9, N10, P10	N9, P9, N10, P10
PST[3:0]	—	—	—	O	EVDD	N8, P8, L9, M9	R10, T10, R11, T11	R10, T10, R11, T11
Test								
TEST[7]	—	—	—	I	EVDD	E10	A16	A16
PLL_TEST[8]	—	—	—	I	EVDD	—	N13	N13
Power Supplies								
EVDD	—	—	—	—	—	E6, E7, F5–F7, H9, J8, J9, K8, K9, K11	E8, F5–F8, G5, G6, H5, H6, J11, K11, K12, L9–L11, M9, M10	E8, F5–F8, G5, G6, H5, H6, J11, K11, K12, L9–L11, M9, M10
IVDD	—	—	—	—	—	E5, K5, K10, J10	E5, G12, M5, M11, M12	E5, G12, M5, M11, M12
PLL_VDD	—	—	—	—	—	H10	J12	J12
SD_VDD	—	—	—	—	—	E8, E9, F8–F10, J5–J7, K7	E9, F9–F11, G11, H11, J5, J6, K5, K6, L5–L8, M6, M7	E9, F9–F11, G11, H11, J5, J6, K5, K6, L5–L8, M6, M7

Table 13-1. MCF5327/8/9 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir.[1]	Voltage Domain	MCF5327 196 MAPBGA	MCF5328 256 MAPBGA	MCF53281 MCF5329 256 MAPBGA
USB_VDD	—	—	—	—	—	G10	L14	L14
VSS	—	—	—	—	—	G6–G9, H6–H8, P9	G7–G10, H7–H10, J7–10, K7–K10, L12, L13	G7–G10, H7–H10, J7–10, K7–K10, L12, L13
PLL_VSS	—	—	—	—	—	H11	K13	K13
USB_VSS	—	—	—	—	—	H12	M14	M14

[1] Refers to pin's primary function.

[2] Pull-up enabled internally on this signal for this mode.

[3] The SDRAM functions of these signals are not programmable by the user. They are dynamically switched by the processor when accessing SDRAM memory space and are included here for completeness.

[4] Primary functionality selected by asserting the DRAMSEL signal (SDR mode). Alternate functionality selected by negating the DRAMSEL signal (DDR mode). The GPIO module is not responsible for assigning these pins.

[5] GPIO functionality is determined by the edge port module. The GPIO module is only responsible for assigning the alternate functions.

[6] If JTAG_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.

[7] Pull-down enabled internally on this signal for this mode.

[8] Must be left floating for proper operation of the PLL.

Refer to the Chapter 2, "Signal Descriptions," for more detailed descriptions of these pins and other pins not controlled by the ports module. The function of most of the pins (primary function, GPIO, etc.) is determined by the ports module pin assignment registers.

It should be noted from Table 13-2 that 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 13-2.

Table 13-2. Multiple-Pin Functions

Function	Direction	Associated Pins
$\overline{FB\_CS5}$	O	$\overline{FB\_CS5}$, A23
$\overline{FB\_CS4}$	O	$\overline{FB\_CS5}$, A22
I2C_SDA	I/O	I2C_SDA, QSPI_DOUT, FEC_MDIO
I2C_SCL	I/O	I2C_SCL, QSPI_CLK, FEC_MDC
CANRX	I	LCD_D16, I2C_SDA, SSI_RXD
CANTX	O	LCD_D17, I2C_SCL, SSI_TXD

Table 13-2. Multiple-Pin Functions (continued)

Function	Direction	Associated Pins
U2TXD	O	I2C_SCL, DT2IN, SSI_TXD
U2RXD	I	I2C_SDA, DT3IN, SSI_RXD
$\overline{\text{U2CTS}}$	I	QSPI_DIN, SSI_BCLK
$\overline{\text{U2RTS}}$	O	SSI_FS, QSPI_PCS2
DT2OUT	O	DT2IN, PWM1
DT3OUT	O	DT3IN, PWM3
DT2IN	I	DT2IN, PWM1
DT3IN	I	DT3IN, PWM3
PWM7	O	QSPI_PCS1, SSI_BCLK, PWM7
PWM5	O	QSPI_PCS0, SSI_FS, PWM5
SSI_MCLK	O	SSI_MCLK, IRQ3
SSI_BCLK	I	SSI_BCLK, $\overline{\text{U1CTS}}$
SSI_FS	I/O	SSI_FS, $\overline{\text{U1RTS}}$
SSI_RXD	I	SSI_RXD, U1RXD
SSI_TXD	O	SSI_TXD, U1TXD

13.3   Memory Map/Register Definition

Table 13-3 summarizes all the registers in the ports address space.

Table 13-3. GPIO Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Port Output Data Registers					
0xFC0A_4000	PODR_FECH	8	R/W	0xFF	13.3.1/13-14
0xFC0A_4001	PODR_FECL	8	R/W	0xFF	13.3.1/13-14
0xFC0A_4002	PODR_SSI	8	R/W	0x1F	13.3.1/13-14
0xFC0A_4003	PODR_BUSCTL	8	R/W	0x0F	13.3.1/13-14
0xFC0A_4004	PODR_BE	8	R/W	0x0F	13.3.1/13-14
0xFC0A_4005	PODR_CS	8	R/W	0x3E	13.3.1/13-14
0xFC0A_4006	PODR_PWM	8	R/W	0x3C	13.3.1/13-14
0xFC0A_4007	PODR_FECI2C	8	R/W	0x0F	13.3.1/13-14
0xFC0A_4009	PODR_UART	8	R/W	0xFF	13.3.1/13-14
0xFC0A_400A	PODR_QSPI	8	R/W	0x3F	13.3.1/13-14

Table 13-3. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_400B	PODR_TIMER	8	R/W	0x0F	13.3.1/13-14
0xFC0A_400D	PODR_LCDDATAH	8	R/W	0x03	13.3.1/13-14
0xFC0A_400E	PODR_LCDDATAM	8	R/W	0xFF	13.3.1/13-14
0xFC0A_400F	PODR_LCDDATAL	8	R/W	0xFF	13.3.1/13-14
0xFC0A_4010	PODR_LCDCTLH	8	R/W	0x01	13.3.1/13-14
0xFC0A_4011	PODR_LCDCTLL	8	R/W	0xFF	13.3.1/13-14
Port Data Direction Registers					
0xFC0A_4014	PDDR_FECH	8	R/W	0x00	13.3.2/13-17
0xFC0A_4015	PDDR_FECL	8	R/W	0x00	13.3.2/13-17
0xFC0A_4016	PDDR_SSI	8	R/W	0x00	13.3.2/13-17
0xFC0A_4017	PDDR_BUSCTL	8	R/W	0x00	13.3.2/13-17
0xFC0A_4018	PDDR_BE	8	R/W	0x00	13.3.2/13-17
0xFC0A_4019	PDDR_CS	8	R/W	0x00	13.3.2/13-17
0xFC0A_401A	PDDR_PWM	8	R/W	0x00	13.3.2/13-17
0xFC0A_401B	PDDR_FECI2C	8	R/W	0x00	13.3.2/13-17
0xFC0A_401D	PDDR_UART	8	R/W	0x00	13.3.2/13-17
0xFC0A_401E	PDDR_QSPI	8	R/W	0x00	13.3.2/13-17
0xFC0A_401F	PDDR_TIMER	8	R/W	0x00	13.3.2/13-17
0xFC0A_4021	PDDR_LCDDATAH	8	R/W	0x00	13.3.2/13-17
0xFC0A_4022	PDDR_LCDDATAM	8	R/W	0x00	13.3.2/13-17
0xFC0A_4023	PDDR_LCDDATAL	8	R/W	0x00	13.3.2/13-17
0xFC0A_4024	PDDR_LCDCTLH	8	R/W	0x00	13.3.2/13-17
0xFC0A_4025	PDDR_LCDCTLL	8	R/W	0x00	13.3.2/13-17
Port Pin Data/Set Data Registers					
0xFC0A_4028	PPDSDR_FECH	8	R/W	See Section	13.3.3/13-19
0xFC0A_4029	PPDSDR_FECL	8	R/W	See Section	13.3.3/13-19
0xFC0A_402A	PPDSDR_SSI	8	R/W	See Section	13.3.3/13-19
0xFC0A_402B	PPDSDR_BUSCTL	8	R/W	See Section	13.3.3/13-19
0xFC0A_402C	PPDSDR_BE	8	R/W	See Section	13.3.3/13-19
0xFC0A_402D	PPDSDR_CS	8	R/W	See Section	13.3.3/13-19
0xFC0A_402E	PPDSDR_PWM	8	R/W	See Section	13.3.3/13-19
0xFC0A_402F	PPDSDR_FECI2C	8	R/W	See Section	13.3.3/13-19

Table 13-3. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4031	PPDSDR_UART	8	R/W	See Section	13.3.3/13-19
0xFC0A_4032	PPDSDR_QSPI	8	R/W	See Section	13.3.3/13-19
0xFC0A_4033	PPDSDR_TIMER	8	R/W	See Section	13.3.3/13-19
0xFC0A_4035	PPDSDR_LCDDATAH	8	R/W	See Section	13.3.3/13-19
0xFC0A_4036	PPDSDR_LCDDATAM	8	R/W	See Section	13.3.3/13-19
0xFC0A_4037	PPDSDR_LCDDATAL	8	R/W	See Section	13.3.3/13-19
0xFC0A_4038	PPDSDR_LCDCTLH	8	R/W	See Section	13.3.3/13-19
0xFC0A_4039	PPDSDR_LCDCTLL	8	R/W	See Section	13.3.3/13-19
Port Clear Output Data Registers					
0xFC0A_403C	PCLRR_FECH	8	W	0x00	13.3.4/13-21
0xFC0A_403D	PCLRR_FECL	8	W	0x00	13.3.4/13-21
0xFC0A_403E	PCLRR_SSI	8	W	0x00	13.3.4/13-21
0xFC0A_403F	PCLRR_BUSCTL	8	W	0x00	13.3.4/13-21
0xFC0A_4040	PCLRR_BE	8	W	0x00	13.3.4/13-21
0xFC0A_4041	PCLRR_CS	8	W	0x00	13.3.4/13-21
0xFC0A_4042	PCLRR_PWM	8	W	0x00	13.3.4/13-21
0xFC0A_4043	PCLRR_FECI2C	8	W	0x00	13.3.4/13-21
0xFC0A_4045	PCLRR_UART	8	W	0x00	13.3.4/13-21
0xFC0A_4046	PCLRR_QSPI	8	W	0x00	13.3.4/13-21
0xFC0A_4047	PCLRR_TIMER	8	W	0x00	13.3.4/13-21
0xFC0A_4049	PCLRR_LCDDATAH	8	W	0x00	13.3.4/13-21
0xFC0A_404A	PCLRR_LCDDATAM	8	W	0x00	13.3.4/13-21
0xFC0A_404B	PCLRR_LCDDATAL	8	W	0x00	13.3.4/13-21
0xFC0A_404C	PCLRR_LCDCTLH	8	W	0x00	13.3.4/13-21
0xFC0A_404D	PCLRR_LCDCTLL	8	W	0x00	13.3.4/13-21
Pin Assignment Registers					
0xFC0A_4050	PAR_FEC	8	R/W	0x00	13.3.5.8/13-29
0xFC0A_4051	PAR_PWM	8	R/W	0x00	13.3.5.11/13-31
0xFC0A_4052	PAR_BUSCTL	8	R/W	0xF8	13.3.5.1/13-24
0xFC0A_4053	PAR_FECI2C	8	R/W	0x00	13.3.5.4/13-26
0xFC0A_4054	PAR_BE	8	R/W	0x0F	13.3.5.2/13-24
0xFC0A_4055	PAR_CS	8	R/W	0x3E	13.3.5.3/13-25

Table 13-3. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4056	PAR_SSI	8	R/W	0x0000	13.3.5.10/13-30
0xFC0A_4058	PAR_UART	8	R/W	0x0000	13.3.5.7/13-28
0xFC0A_405A	PAR_QSPI	8	R/W	0x0000	13.3.5.5/13-26
0xFC0A_405C	PAR_TIMER	8	R/W	0x00	13.3.5.6/13-27
0xFC0A_405D	PAR_LCDDATA	8	R/W	0x00	13.3.5.12/13-32
0xFC0A_405E	PAR_LCDCTL	16	R/W	0x0000	13.3.5.13/13-33
0xFC0A_4060	PAR_IRQ	16	R/W	0x0000	13.3.5.9/13-30
Mode Select Control Registers					
0xFC0A_4064	MSCR_FLEXBUS	8	R/W	0x3F	13.3.6/13-34
0xFC0A_4065	MSCR_SDRAM	8	R/W	0x3F	13.3.7/13-34
Drive Strength Control Registers					
0xFC0A_4068	DSCR_I2C	8	R/W	See Section	13.3.8/13-35
0xFC0A_4069	DSCR_PWM	8	R/W	See Section	13.3.8/13-35
0xFC0A_406A	DSCR_FEC	8	R/W	See Section	13.3.8/13-35
0xFC0A_406B	DSCR_UART	8	R/W	See Section	13.3.8.1/13-36
0xFC0A_406C	DSCR_QSPI	8	R/W	See Section	13.3.8/13-35
0xFC0A_406D	DSCR_TIMER	8	R/W	See Section	13.3.8/13-35
0xFC0A_406E	DSCR_SSI	8	R/W	See Section	13.3.8/13-35
0xFC0A_406F	DSCR_LCD	8	R/W	See Section	13.3.8/13-35
0xFC0A_4070	DSCR_DEBUG	8	R/W	See Section	13.3.8/13-35
0xFC0A_4071	DSCR_CLKRST	8	R/W	See Section	13.3.8.2/13-37
0xFC0A_4072	DSCR_IRQ	8	R/W	See Section	13.3.8/13-35

13.3.1 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_4003 (PODR_BUSCTL)   Access: User read/write
        0xFC0A_4004 (PODR_BE)
        0xFC0A_4007 (PODR_FECI2C)
        0xFC0A_400B (PODR_TIMER)

	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
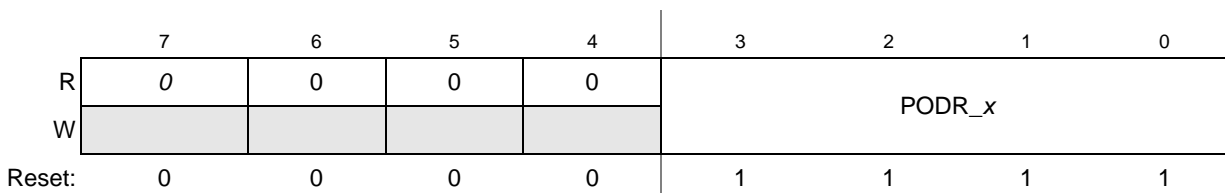
Figure 13-2. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4000 (PODR_FECH)   Access: User read/write
        0xFC0A_4001 (PODR_FECL)
        0xFC0A_4009 (PODR_UART)
        0xFC0A_400E (PODR_LCDDATAM)
        0xFC0A_400F (PODR_LCDDATAL)
        0xFC0A_4011 (PODR_LCDCTLL)

	7	6	5	4	3	2	1	0
R				PODR_*x*				
W								
Reset:	1	1	1	1	1	1	1	1
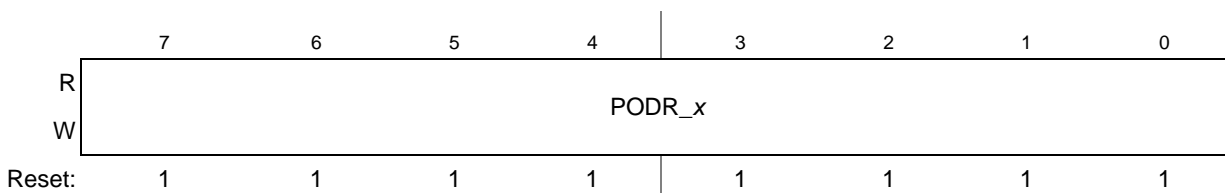
Figure 13-3. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4006 (PODR_PWM)   Access: User read/write

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

Figure 13-4. Port PWM Output Data Register (PODR_PWM)

Address: 0xFC0A_400A (PODR_QSPI)   Access: User read/write

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

Figure 13-5. Port QSPI Output Data Register (PODR_QSPI)

Address: 0xFC0A_4005 (PODR_CS)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0			PODR_CS			0
W								
Reset:	0	0	1	1	1	1	1	0
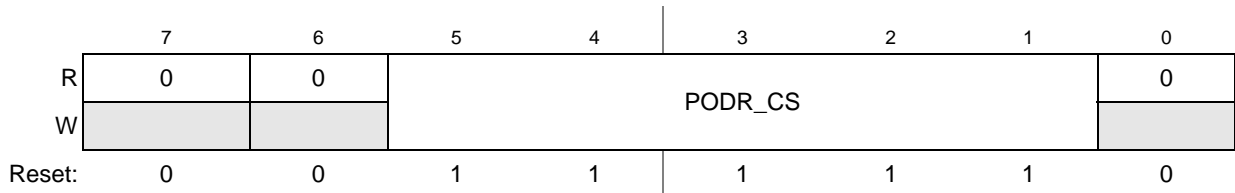
Figure 13-6. Port CS Output Data Register (PODR_CS)

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
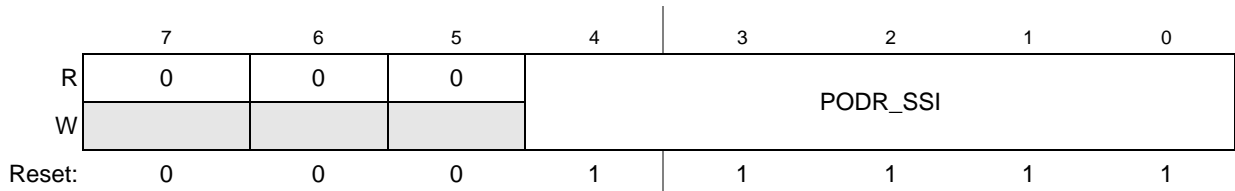
Figure 13-7. Port SSI Output Data Register (PODR_SSI)

Address: 0xFC0A_400D (PODR_LCDDATAH)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PODR_LCDDATAH	
W								
Reset:	0	0	0	0	0	0	1	1
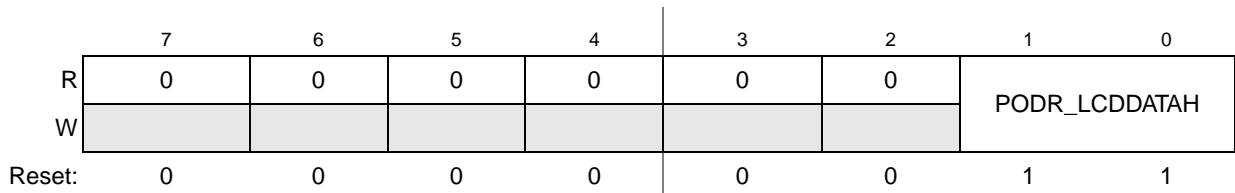
Figure 13-8. Port LCDDATAH Output Data Register (PODR_LCDDATAH)

Address: 0xFC0A_4010 (PODR_LCDCTLH)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PODR_ LCDCTLH
W								
Reset:	0	0	0	0	0	0	0	1
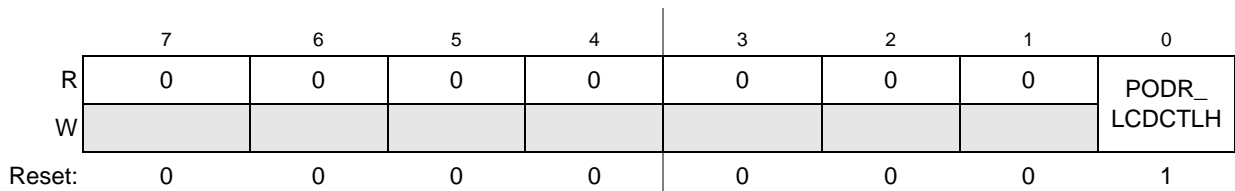
Figure 13-9. Port LCDCTLH Output Data Register (PODR_LCDCTLH)

Table 13-4. 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.

13.3.2 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_4017 (PDDR_BUSCTL)   Access: User read/write
         0xFC0A_4018 (PDDR_BE)
         0xFC0A_401B (PDDR_FECI2C)
         0xFC0A_401F (PDDR_TIMER)

	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
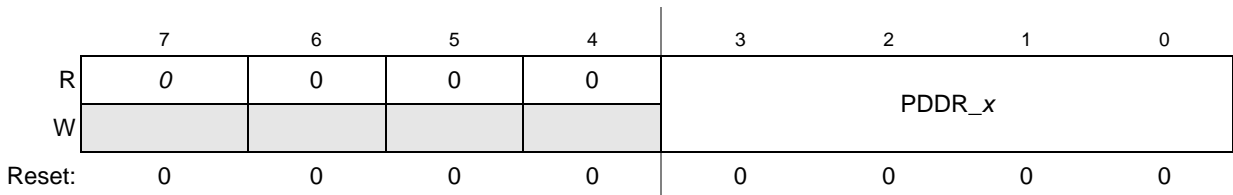
Figure 13-10. Port Data Direction Registers (PDDR_*x*)

Address: 0xFC0A_4014 (PDDR_FECH)   Access: User read/write
         0xFC0A_4015 (PDDR_FECL)
         0xFC0A_401D (PDDR_UART)
         0xFC0A_4022 (PDDR_LCDDATAM)
         0xFC0A_4023 (PDDR_LCDDATAL)
         0xFC0A_4025 (PDDR_LCDCTLL)

	7	6	5	4	3	2	1	0
R				PDDR_*x*				
W								
Reset:	0	0	0	0	0	0	0	0
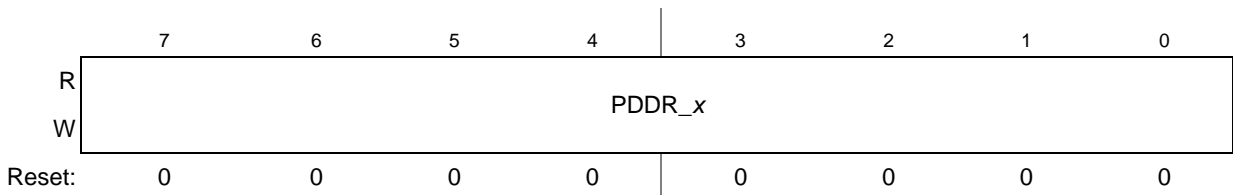
Figure 13-11. Port Data Direction Registers (PDDR_*x*)

Address: 0xFC0A_401A (PDDR_PWM)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0		PDDR_PWM			0	0
W								
Reset:	0	0	0	0	0	0	0	0
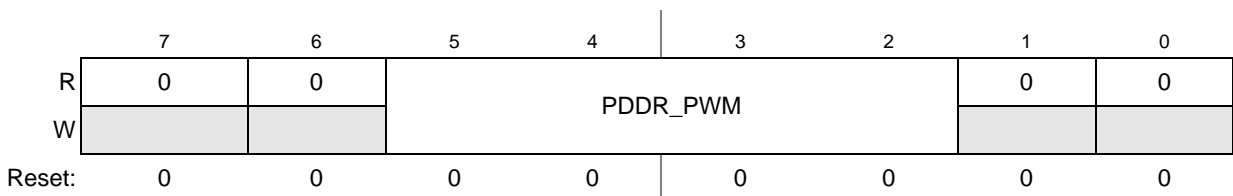
Figure 13-12. Port PWM Data Direction Register (PDDR_PWM)

Address: 0xFC0A_401E (PDDR_QSPI)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0			PDDR_QSPI			
W								
Reset:	0	0	0	0	0	0	0	0
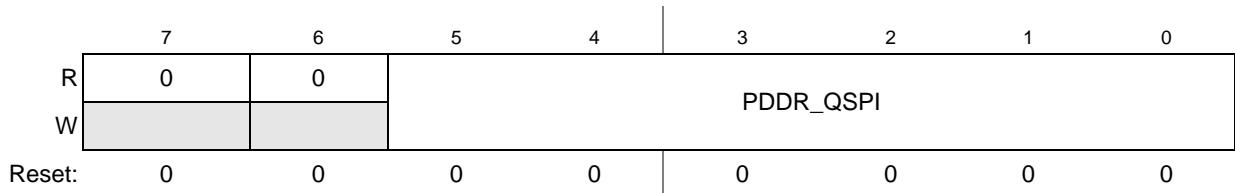
Figure 13-13. Port QSPI Data Direction Register (PDDR_QSPI)

Address: 0xFC0A_4019 (PDDR_CS)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0		PDDR_CS				0
W								
Reset:	0	0	0	0	0	0	0	0
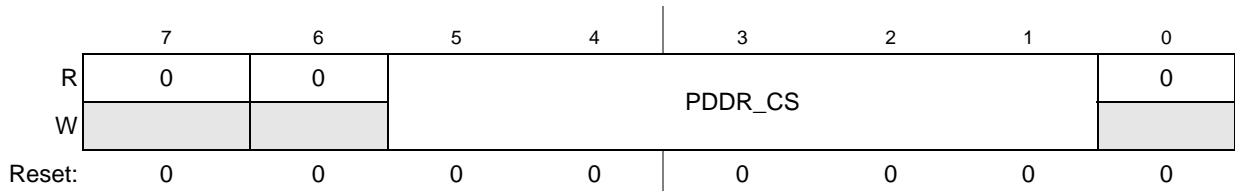
Figure 13-14. Port CS Data Direction Register (PDDR_CS)

Address: 0xFC0A_4016 (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
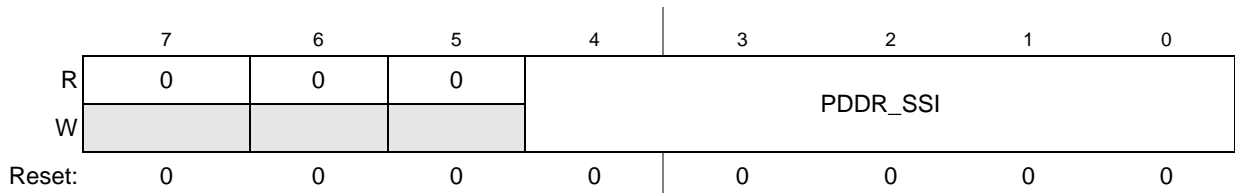
Figure 13-15. Port SSI Data Direction Register (PDDR_SSI)

Address: 0xFC0A_4021 (PDDR_LCDDATAH)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PDDR_LCDDATAH	
W								
Reset:	0	0	0	0	0	0	0	0
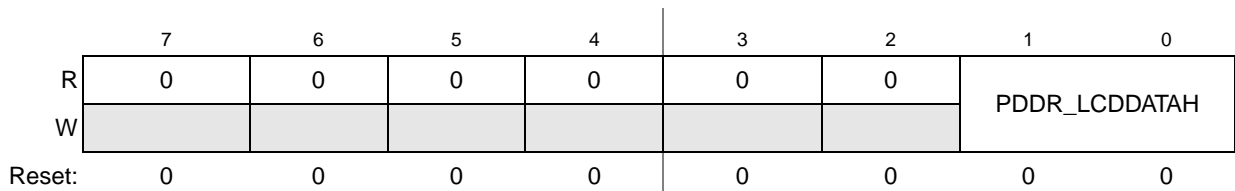
Figure 13-16. Port LCDDATAH Data Direction Register (PDDR_LCDDATAH)

Address: 0xFC0A_4024 (PDDR_LCDCTLH)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PDDR_LCDCTLH
W								
Reset:	0	0	0	0	0	0	0	0
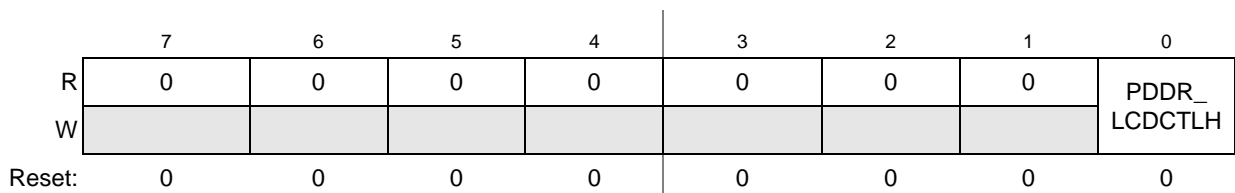
Figure 13-17. Port LCDCTLH Data Direction Register (PDDR_LCDCTLH)

Table 13-5. 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.

13.3.3 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 0s has no effect.

Address: 0xFC0A_402B (PPDSDR_BUSCTL)   Access: User read/write
         0xFC0A_402C (PPDSDR_BE)
         0xFC0A_402F (PPDSDR_FECI2C)
         0xFC0A_4033 (PPDSDR_TIMER)

	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]
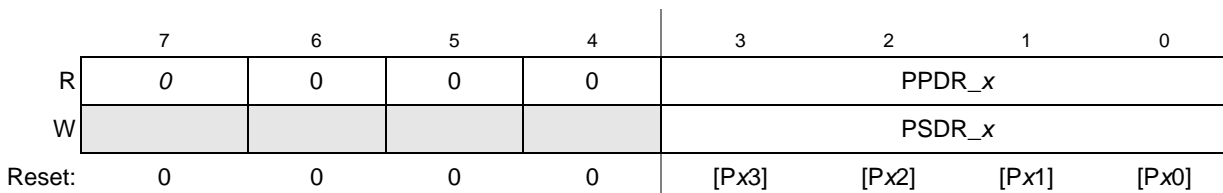
Figure 13-18. Port Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_4028 (PPDSDR_FECH)   Access: User read/write
         0xFC0A_4029 (PPDSDR_FECL)
         0xFC0A_4031 (PPDSDR_UART)
         0xFC0A_4036 (PPDSDR_LCDDATAM)
         0xFC0A_4037 (PPDSDR_LCDDATAL)
         0xFC0A_4039 (PPDSDR_LCDCTLL)

	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 13-19. Port Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_402E (PPDSDR_PWM)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0			PPDSDR_PWM			0	0
W									
Reset:	0	0	[PPWM5]	[PPWM4]	[PPWM3]	[PPWM2]	0	0
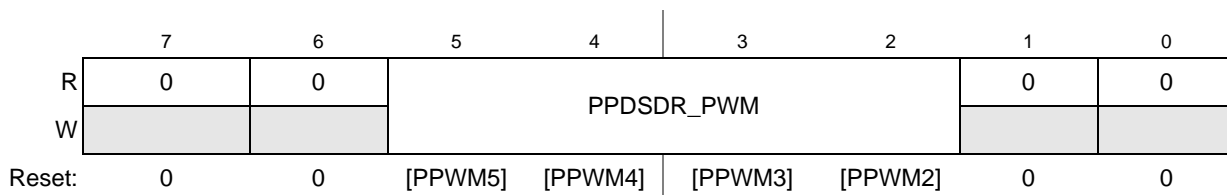
Figure 13-20. Port PWM Pin Data/Set Data Register (PPDSDR_PWM)

Address: 0xFC0A_4032 (PPDSDR_QSPI)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0			PPDSDR_QSPI			
W								
Reset:	0	0	[PQSPI5]	[PQSPI4]	[PQSPI3]	[PQSPI2]	[PQSPI1]	[PQSPI0]
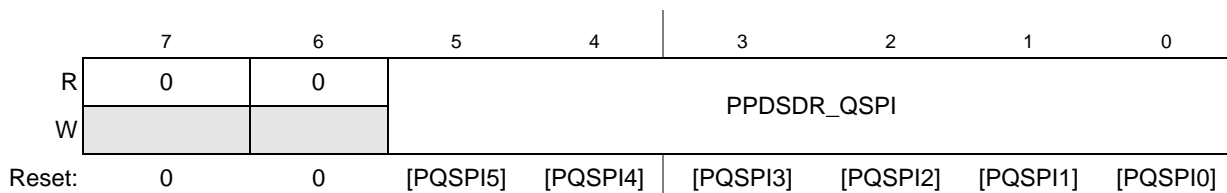
Figure 13-21. Port QSPI Pin Data/Set Data Register (PPDSDR_QSPI)

Address: 0xFC0A_402D (PPDSDR_CS)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0			PPDSDR_CS			0
W								
Reset:	0	0	[PCS3]	[PCS4]	[PCS3]	[PCS2]	[PCS1]	0
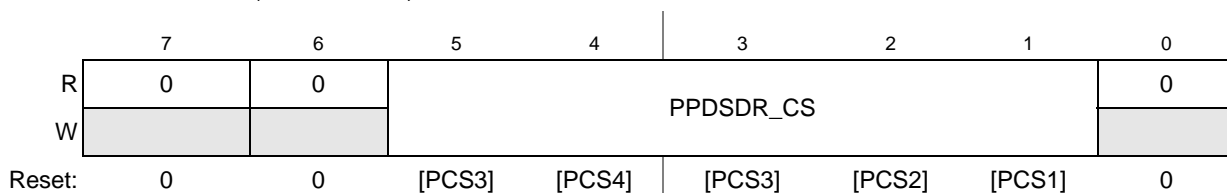
Figure 13-22. Port CS Pin Data/Set Data Register (PPDSDR_CS)

Address: 0xFC0A_402A (PPDSDR_SSI)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0		PPDSDR_SSI			
W								
Reset:	0	0	0	[PSSI4]	[PSSI3]	[PSSI2]	[PSSI1]	[PSSI0]
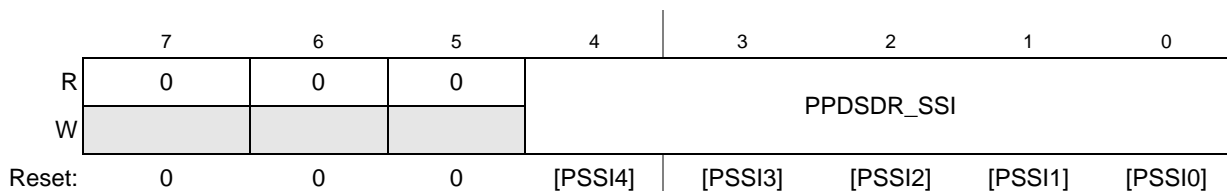
Figure 13-23. Port SSI Pin Data/Set Data Register (PPDSDR_SSI)

Address: 0xFC0A_4035 (PPDSDR_LCDDATAH)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PPDSDR_LCDDATAH	
W								
Reset:	0	0	0	0	0	0	[PLCD DATAH1]	[PLCD DATAH0]
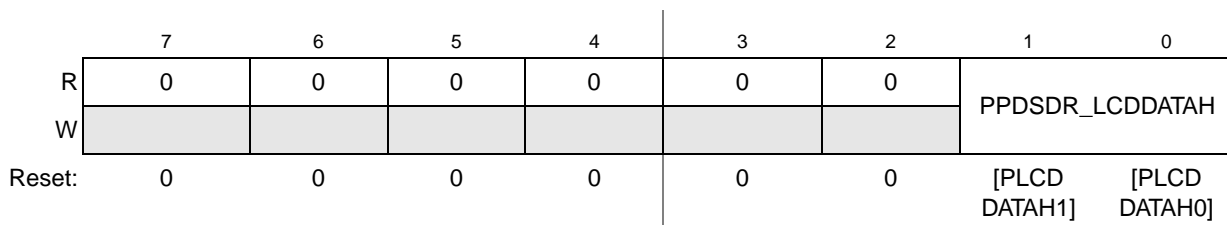
Figure 13-24. Port LCDDATAH Pin Data/Set Data Register (PPDSDR_LCDDATAH)

Address: 0xFC0A_4038 (PPDSDR_LCDCTLH)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PPDSDR_
W								LCDCTLH0
Reset:	0	0	0	0	0	0	0	[PLCD CTLH0]
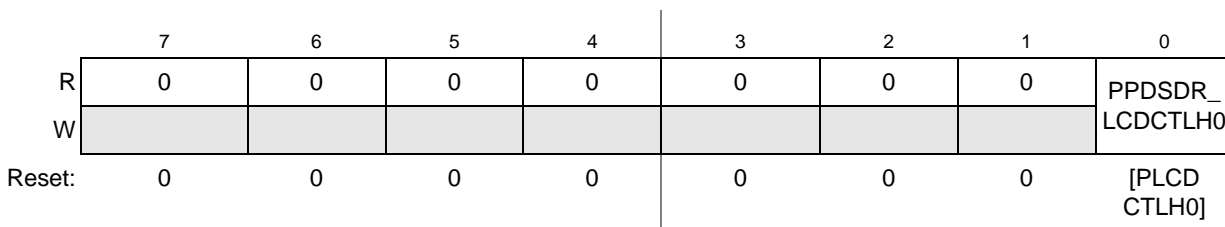
Figure 13-25. Port LCDCTLH Pin Data/Set Data Register (PPDSDR_LCDCTLH)

Table 13-6. 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
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.

13.3.4 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 0s. 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_403F (PCLRR_BUSCTL)   Access: User write-only
0xFC0A_4040 (PCLRR_BE)
0xFC0A_4043 (PCLRR_FECI2C)
0xFC0A_4047 (PCLRR_TIMER)

	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
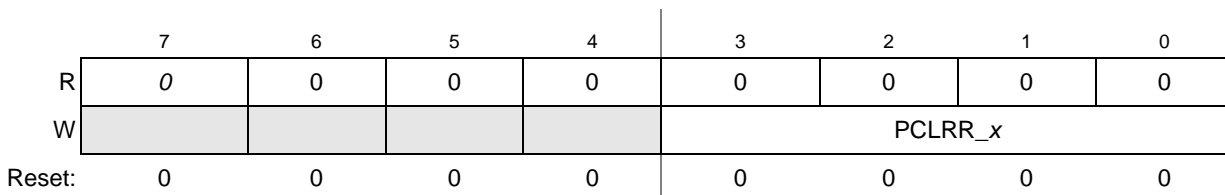
Figure 13-26. Port Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_403C (PCLRR_FECH)   Access: User write-only
         0xFC0A_403D (PCLRR_FECL)
         0xFC0A_4045 (PCLRR_UART)
         0xFC0A_404A (PCLRR_LCDDATAM)
         0xFC0A_404B (PCLRR_LCDDATAL)
         0xFC0A_404D (PCLRR_LCDCTLL)

	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
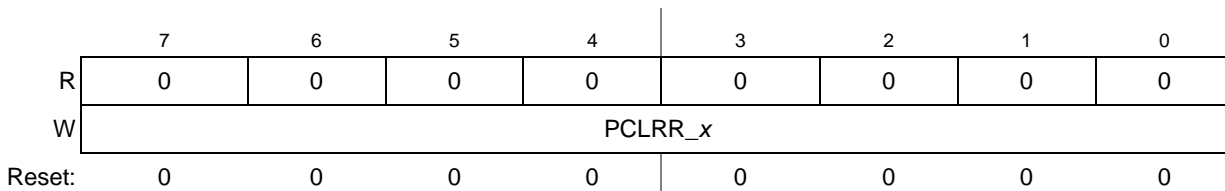
Figure 13-27. Port Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_4042 (PCLRR_PWM)   Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W				PCLRR_PWM				
Reset:	0	0	0	0	0	0	0	0
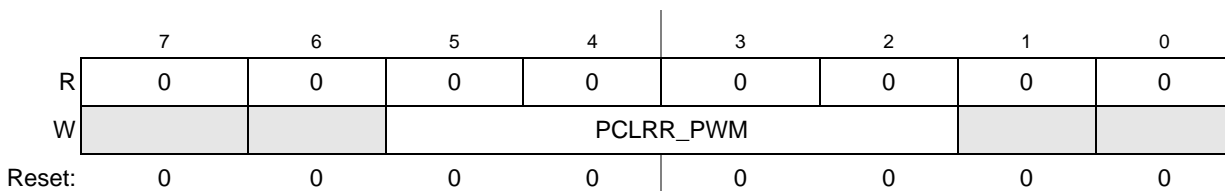
Figure 13-28. Port PWM Clear Output Data Register (PCLRR_PWM)

Address: 0xFC0A_4046 (PCLRR_QSPI)   Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W					PCLRR_QSPI			
Reset:	0	0	0	0	0	0	0	0
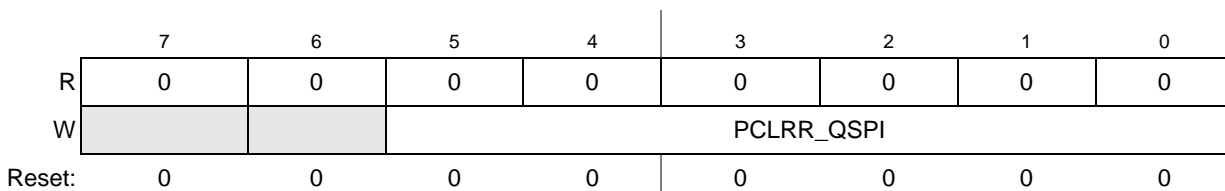
Figure 13-29. Port QSPI Clear Output Data Register (PCLRR_QSPI)

Address: 0xFC0A_4041 (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
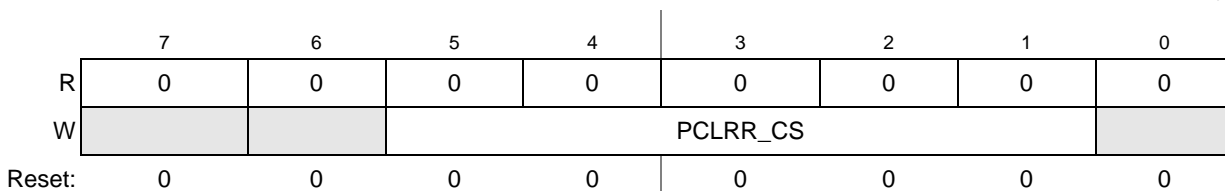
Figure 13-30. Port CS Clear Output Data Register (PCLRR_CS)

Address: 0xFC0A_403E (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
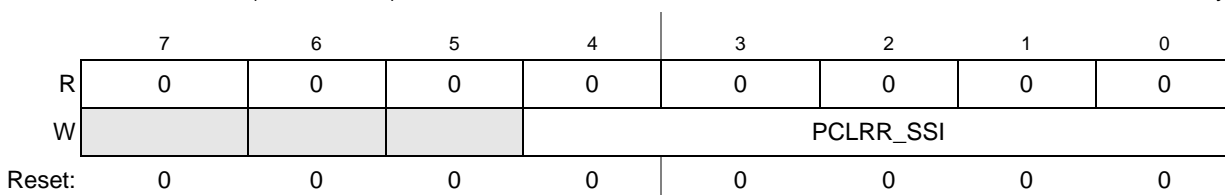
Figure 13-31. Port SSI Clear Output Data Register (PCLRR_SSI)

Address: 0xFC0A_4049 (PCLRR_LCDDATAH)   Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W							PCLRR_LCDDATAH	
Reset:	0	0	0	0	0	0	0	0
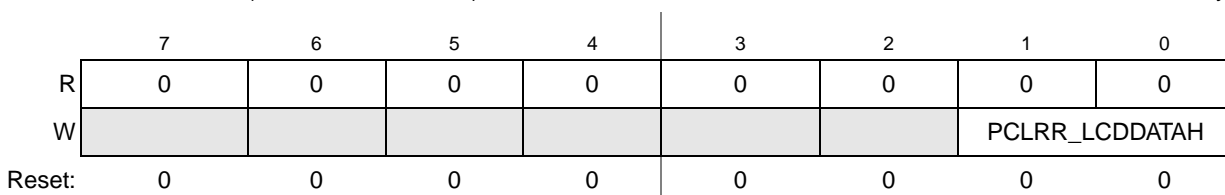
Figure 13-32. Port LCDDATAH Clear Output Data Register (PCLRR_LCDDATAH)

Address: 0xFC0A_404C (PCLRR_LCDCTLH)   Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								PCLRR_ LCDCTLH
Reset:	0	0	0	0	0	0	0	0
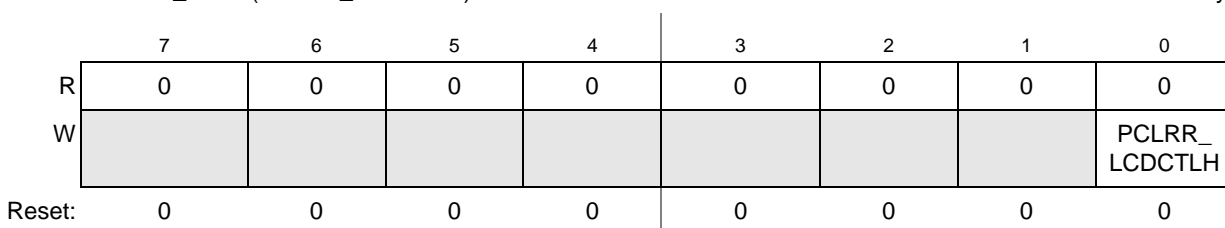
Figure 13-33. Port LCDCTLH Clear Output Data Register (PCLRR_LCDCTLH)

Table 13-7. 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.

13.3.5 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.

13.3.5.1 External Bus Control Pin Assignment Register (PAR_BUSCTL)

The PAR_BUSCTL register controls the functions of the external bus control signal pins.

Address: 0xFC0A_4052 (PAR_BUSCTL)   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	PAR_OE	PAR_TA	PAR_RWB	PAR_TS				
Reset:	1	1	1	1	1	0	0	0
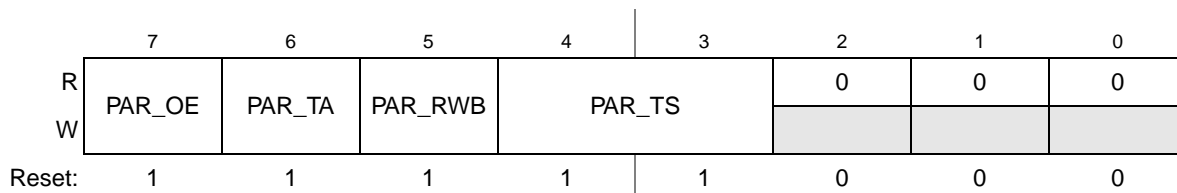
Figure 13-34. External Bus Control Pin Assignment Register (PAR_BUSCTL)

Table 13-8. PAR_BUSCTL Field Descriptions

Field	Description
7 PAR_OE	0 $\overline{\text{OE}}$ pin configured for GPIO 1 $\overline{\text{OE}}$ pin configured for external bus $\overline{\text{OE}}$ function
6 PAR_TA	0 $\overline{\text{TA}}$ pin configured for GPIO 1 $\overline{\text{TA}}$ pin configured for external bus $\overline{\text{TA}}$ function
5 PAR_RWB	0 R/$\overline{\text{W}}$ pin configured for GPIO 1 R/$\overline{\text{W}}$ pin configured for external bus read/write function
4–3 PAR_TS	00 $\overline{\text{TS}}$ pin configured for GPIO 01 Reserved 10 $\overline{\text{TS}}$ pin configured for DMA acknowledge 0 function 11 $\overline{\text{TS}}$ pin configured for external bus $\overline{\text{TS}}$ function
2–0	Reserved, should be cleared.

13.3.5.2 Byte Enable Pin Assignment Register (PAR_BE)

The PAR_BE register controls the functions of the byte enable pins.

Address: 0xFC0A_4054 (PAR_BE)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_BE			
W					PAR_BE			
Reset:	0	0	0	0	1	1	1	1
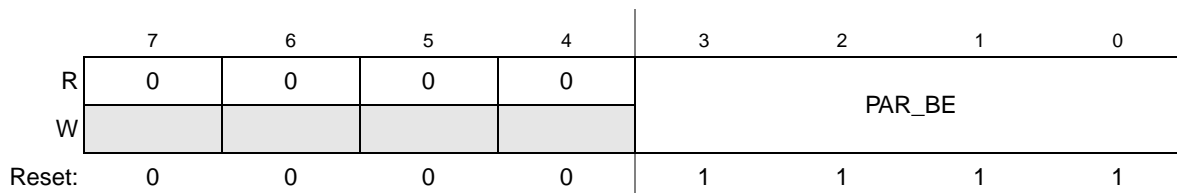
Figure 13-35. Byte Enable Pin Assignment Register (PAR_BE)

Table 13-9. PAR_BE Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 PAR_BE	$\overline{BE/BWE}$[3:0] pin assignment. The PAR_BE[3:0] bits configure the $\overline{BE/BWE}$[3:0] pins for their primary function or GPIO. 0 $\overline{BE/BWE}$[3:0] pin configured for GPIO 1 $\overline{BE/BWE}$[3:0] pin configured for $\overline{BE/BWE}$[3:0] function Refer to Chapter 9, "Chip Configuration Module (CCM)" for more information on reset configuration.

13.3.5.3   Chip Select Pin Assignment Register (PAR_CS)

The PAR_CS register controls the functions of the FlexBus chip select pins.

Address: 0xFC0A_4055 (PAR_CS)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PAR_CS5	PAR_CS4	PAR_CS3	PAR_CS2	PAR_CS1	0
W								
Reset:	0	0	1	1	1	1	1	0
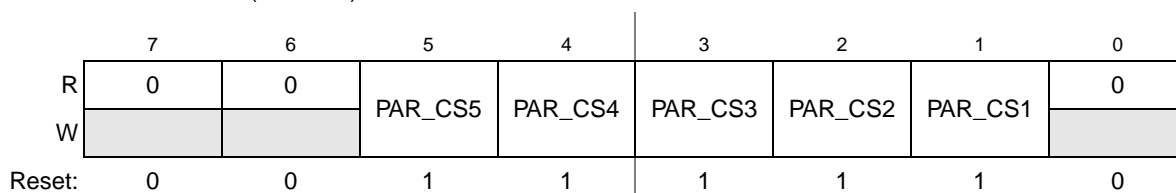
Figure 13-36. Chip Select Pin Assignment Register (PAR_CS)

Table 13-10. PAR_CS Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 PAR_CS5	$\overline{FB\_CS5}$ pin assignment. 0 $\overline{FB\_CS5}$ pin configured for GPIO 1 $\overline{FB\_CS5}$ pin configured for FlexBus $\overline{FB\_CS5}$ function
4 PAR_CS4	$\overline{FB\_CS4}$ pin assignment. 0 $\overline{FB\_CS4}$ pin configured for GPIO 1 $\overline{FB\_CS4}$ pin configured for FlexBus $\overline{FB\_CS4}$ function
3 PAR_CS3	$\overline{FB\_CS3}$ pin assignment. 0 $\overline{FB\_CS3}$ pin configured for GPIO 1 $\overline{FB\_CS3}$ pin configured for FlexBus $\overline{FB\_CS3}$ function
2 PAR_CS2	$\overline{FB\_CS2}$ pin assignment. 0 $\overline{FB\_CS2}$ pin configured for GPIO 1 $\overline{FB\_CS2}$ pin configured for FlexBus $\overline{FB\_CS2}$ function
1 PAR_CS1	$\overline{FB\_CS1}$ pin assignment. 0 $\overline{FB\_CS1}$ pin configured for GPIO 1 $\overline{FB\_CS1}$ pin configured for FlexBus $\overline{FB\_CS1}$ function
0	Reserved, should be cleared.

13.3.5.4   FEC/I2C Pin Assignment Register (PAR_FECI2C)

The PAR_FECI2C register controls the functions of the I^{2}C and some of the FEC pins.

Address: 0xFC0A_4053 (PAR_FECI2C)   Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_MDC		PAR_MDIO		PAR_SCL		PAR_SDA	
W								
Reset:	0	0	0	0	0	0	0	0
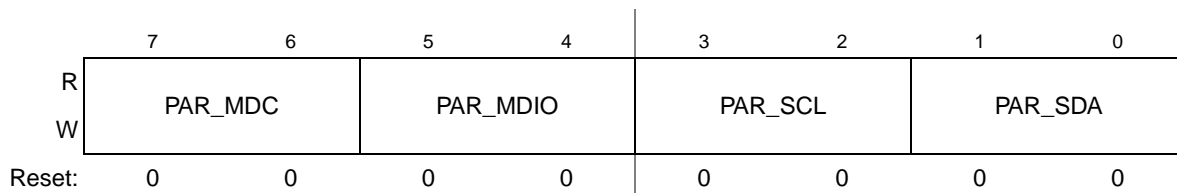
Figure 13-37. FEC/I2C Pin Assignment (PAR_FECI2C)

Table 13-11. PAR_FECI2C Field Descriptions

Field	Description
7–6 PAR_MDC 5–4 PAR_MDIO 3–2 PAR_SCL 1–0 PAR_SDA	FEC & I^{2}C pin assignment. These bit fields configure the FEC_MDC, FEC_MDIO, I2C_SCL, and I2C_SDA pins for one of their primary functions or GPIO. 			PAR_MDC	PAR_MDIO	PAR_SCL	PAR_SDA	

Expanded table inside description:

	PAR_MDC	PAR_MDIO	PAR_SCL	PAR_SDA
00	GPIO	GPIO	GPIO	GPIO
01	Reserved	Reserved	U2TXD	U2RXD
10	I2C_SCL	I2C_SDA	CANTX	CANRX
11	FEC_MDC	FEC_MDIO	I2C_SCL	I2C_SDA

13.3.5.5   QSPI Pin Assignment Register (PAR_QSPI)

The PAR_QSPI register controls the functions of the QSPI pins.

Address: 0xFC0A_405A (PAR_QSPI)   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PAR_PCS2		PAR_PCS1		PAR_PCS0		PAR_DIN		PAR_DOUT		PAR_SCK		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
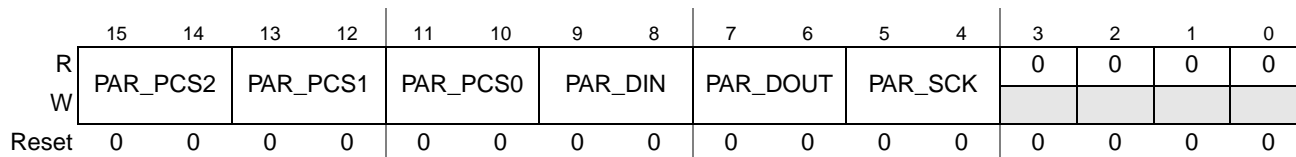
Figure 13-38. QSPI Pin Assignment (PAR_QSPI)

Table 13-12. PAR_QSPI Field Descriptions

Field	Description
15–14 PAR_PCS2 13–12 PAR_PCS1 11–10 PAR_PCS0 9–8 PAR_DIN 7–6 PAR_DOUT 5–4 PAR_SCK	QSPI pin assignment. These bit fields configure the QSPI pins for one of their primary functions or GPIO.

	PAR_PCS2	PAR_PCS1	PAR_PCS0	PAR_DIN	PAR_DOUT	PAR_SCK
00	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO
01	Reserved	USBD_PU_EN	Reserved	Reserved	Reserved	Reserved
10	$\overline{\text{U2RTS}}$	PWM7	PWM5	$\overline{\text{U2CTS}}$	I2C_SDA	I2C_SCL
11	QSPI_PCS2	QSPI_PCS1	QSPI_PCS0	QSPI_DIN	QSPI_DOUT	QSPI_SCK

13.3.5.6 Timer Pin Assignment Registers (PAR_TIMER)

The PAR_TIMER register controls the functions of the DMA timer pins.

Address: 0xFC0A_405C (PAR_TIMER)   Access: User read/write
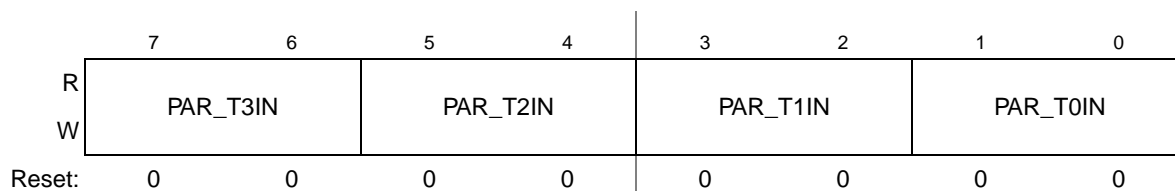
Figure 13-39. Timer Pin Assignment (PAR_TIMER)

Table 13-13. PAR_TIMER Field Descriptions

Field	Description
7–6 PAR_T3IN 5–4 PAR_T2IN 3–2 PAR_T1IN 1–0 PAR_T0IN	DMA Timer pin assignment. These bit fields configure the DMA Timer pins for one of their primary functions or GPIO.

	PAR_T3IN	PAR_T2IN	PAR_T1IN	PAR_T0IN
00	GPIO	GPIO	GPIO	GPIO
01	U2RXD	U2TXD	$\overline{\text{DACK1}}$	$\overline{\text{DREQ0}}$
10	T3OUT	T2OUT	T1OUT	T0OUT
11	T3IN	T2IN	T1IN	T0IN

13.3.5.7   UART Pin Assignment Register (PAR_UART)

The PAR_UART register controls the functions of the UART pins.

Address: 0xFC0A_4058 (PAR_UART)   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_U1CTS		PAR_U1RTS		PAR_U1RXD		PAR_U1TXD		PAR_U0CTS	PAR_U0RTS	PAR_U0RXD	PAR_U0TXD
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-40. UART Pin Assignment (PAR_UART)

Table 13-14. PAR_UART Field Descriptions

Field	Description
15–12	Reserved, should be cleared.
11–10 PAR_U1CTS 9–8 PAR_U1RTS 7–6 PAR_U1RXD 5–4 PAR_U1TXD	UART1 control pin assignment. These bit fields configure the UART1 pins for one of their primary functions or GPIO. <table><tr><td></td><td>**PAR_U1CTS**</td><td>**PAR_U1RTS**</td><td>**PAR_U1RXD**</td><td>**PAR_U1TXD**</td></tr><tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr><tr><td>01</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr><tr><td>10</td><td>SSI_BCLK</td><td>SSI_FS</td><td>SSI_RXD</td><td>SSI_TXD</td></tr><tr><td>11</td><td>$\overline{U1CTS}$</td><td>$\overline{U1RTS}$</td><td>U1RXD</td><td>U1TXD</td></tr></table>
3 PAR_U0CTS	$\overline{U0CTS}$ pin assignment. 0 $\overline{U0CTS}$ pin configured for GPIO 1 $\overline{U0CTS}$ pin configured for UART0 CTS function
2 PAR_U0RTS	$\overline{U0RTS}$ pin assignment. 0 $\overline{U0RTS}$ pin configured for GPIO 1 $\overline{U0RTS}$ pin configured for UART0 RTS function
1 PAR_U0RXD	U0RXD pin assignment. 0 U0RXD pin configured for GPIO 1 U0RXD pin configured for UART0 RXD function
0 PAR_U0TXD	U0TXD pin assignment. 0 U0TXD pin configured for GPIO 1 U0TXD pin configured for UART0 TXD function

13.3.5.8 FEC Pin Assignment Register (PAR_FEC)

The PAR_FEC register controls the functions of the FEC pins.

Address: 0xFC0A_4050 (PAR_FEC)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_FEC_7W		PAR_FEC_MII	
W								
Reset:	0	0	0	0	0	0	0	0
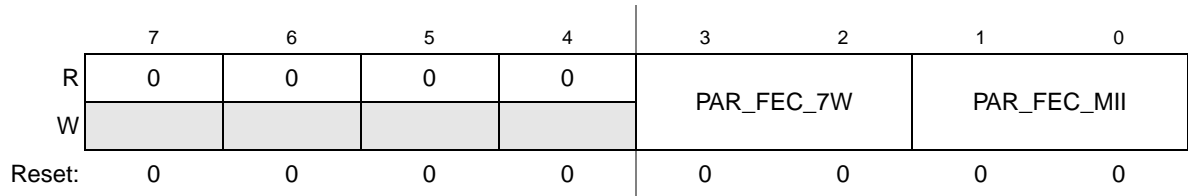
Figure 13-41. FEC Pin Assignment (PAR_FEC)

Table 13-15. PAR_FEC Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–2 PAR_FEC_7W	FEC 7-wire pin assignment. These bit fields configure the FEC_COL, FEC_RXCLK, FEC_RXDV, FEC_RXD0, FEC_TXCLK, FEC_TXD0, and FEC_TXEN pins for one of their primary functions or GPIO. <table><tr><th></th><th>FEC_COL</th><th>FEC_RXCLK</th><th>FEC_RXDV</th><th>FEC_RXD0</th><th>FEC_TXCLK</th><th>FEC_TXD0</th><th>FEC_TXEN</th></tr><tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr><tr><td>01</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr><tr><td>10</td><td>ULPI_CLK</td><td>ULPI_NXT</td><td>ULPI_STP</td><td>ULPI_DATA4</td><td>—</td><td>ULPI_DATA0</td><td>—</td></tr><tr><td>11</td><td>FEC_COL</td><td>FEC_RXCLK</td><td>FEC_RXDV</td><td>FEC_RXD0</td><td>FEC_TXCLK</td><td>FEC_TXD0</td><td>FEC_TXEN</td></tr></table>
1–0 PAR_FEC_MII	FEC MII pin assignment. These bit fields configure the FEC_CRS, FEC_RXD3, FEC_RXD2, FEC_RXD1, FEC_RXER, FEC_TXD3, FEC_TXD2, FEC_TXD1 and FEC_TXER pins for one of their primary functions or GPIO. <table><tr><th></th><th>FEC_RXD3</th><th>FEC_RXD2</th><th>FEC_RXD1</th><th>FEC_RXER</th><th>FEC_CRS</th></tr><tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr><tr><td>01</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr><tr><td>10</td><td>ULPI_DATA7</td><td>ULPI_DATA6</td><td>ULPI_DATA5</td><td>—</td><td>ULPI_DIR</td></tr><tr><td>11</td><td>FEC_RXD3</td><td>FEC_RXD2</td><td>FEC_RXD1</td><td>FEC_RXER</td><td>FEC_CRS</td></tr></table> <table><tr><th></th><th>FEC_TXD3</th><th>FEC_TXD2</th><th>FEC_TXD1</th><th>FEC_TXER</th></tr><tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr><tr><td>01</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr><tr><td>10</td><td>ULPI_DATA3</td><td>ULPI_DATA2</td><td>ULPI_DATA1</td><td>—</td></tr><tr><td>11</td><td>FEC_TXD3</td><td>FEC_TXD2</td><td>FEC_TXD1</td><td>FEC_TXER</td></tr></table> **Note:**

13.3.5.9 IRQ Pin Assignment Register (PAR_IRQ)

The PAR_IRQ register controls the functions of the $\overline{\text{IRQ}}$ pins.

Address: 0xFC0A_4060 (PAR_IRQ)   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	PAR_IRQ6		PAR_IRQ5		PAR_IRQ4		PAR_IRQ2		PAR_IRQ1		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
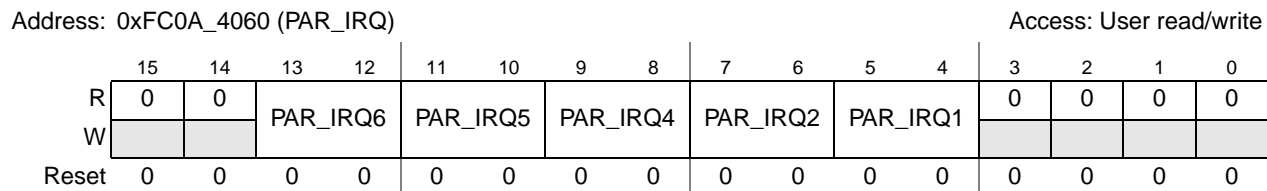
Figure 13-42. IRQ Pin Assignment (PAR_IRQ)

Table 13-16. PAR_IRQ Field Descriptions

Field	Description
15–14	Reserved, should be cleared.
13–12 PAR_IRQ6 11–10 PAR_IRQ5 9–8 PAR_IRQ4 7–6 PAR_IRQ2 5–4 PAR_IRQ1	IRQ pin assignment. These bit fields configure the IRQ pins for one of their primary functions or GPIO. <table><tr><td></td><td>**PAR_IRQ6**</td><td>**PAR_IRQ5**</td><td>**PAR_IRQ4**</td><td>**PAR_IRQ2**</td><td>**PAR_IRQ1I**</td></tr><tr><td>00</td><td>GPIO/$\overline{\text{IRQ6}}$</td><td>GPIO/$\overline{\text{IRQ5}}$</td><td>GPIO/$\overline{\text{IRQ4}}$</td><td>GPIO/$\overline{\text{IRQ2}}$</td><td>GPIO/$\overline{\text{IRQ1}}$</td></tr><tr><td>01</td><td>Reserved</td><td>Reserved</td><td>SSI_MCLK</td><td>Reserved</td><td>SSI_CLKIN</td></tr><tr><td>10</td><td>USBH_VBUS_EN</td><td>USBH_VBUS_OC</td><td>Reserved</td><td>USBH_CLKIN</td><td>$\overline{\text{DREQ1}}$</td></tr><tr><td>11</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>Reserved</td></tr></table>
3–0	Reserved, should be cleared.

13.3.5.10 SSI Pin Assignment Register (PAR_SSI)

The PAR_SSI register controls the functions of the SSI pins.

Address: 0xFC0A_4056 (PAR_SSI)   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PAR_BCLK		PAR_FS		PAR_RXD		PAR_TXD		PAR_MCLK	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
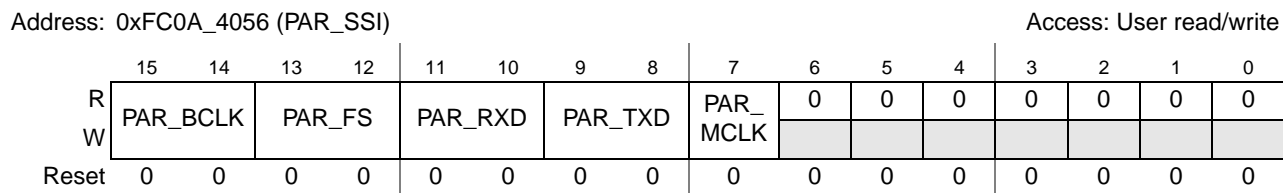
Figure 13-43. SSI Pin Assignment (PAR_SSI)

Table 13-17. PAR_SSI Field Descriptions

Field	Description
15–14 PAR_BCLK 13–12 PAR_FS 11–10 PAR_RXD 9–8 PAR_TXD	SSI pin assignment. These bit fields configure the SSI pins for one of their primary functions or GPIO. |	**PAR_BCLK**	**PAR_FS**	**PAR_RXD**	**PAR_TXD**
---	---	---	---	---
00	GPIO	GPIO	GPIO	GPIO
01	PWM7	PWM5	CANRX	CANTX
10	$\overline{\text{U2CTS}}$	$\overline{\text{U2RTS}}$	U2RXD	U2TXD
11	SSI_BCLK	SSI_FS	SSI_RXD	SSI_TXD	
7 PAR_MCLK	SSI_MCLK pin assignment. 0 SSI_MCLK pin configured for GPIO function. 1 SSI_MCLK pin configured for SSI_MCLK function.
6–0	Reserved, should be cleared.

13.3.5.11 PWM Pin Assignment Register (PAR_PWM)

The PAR_PWM register controls the functions of the PWM output pins.

Address: 0xFC0A_4051 (PAR_PWM)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PAR_PWM7	PAR_PWM5	PAR_PWM3		PAR_PWM1	
W			PAR_PWM7	PAR_PWM5	PAR_PWM3		PAR_PWM1	
Reset:	0	0	0	0	0	0	0	0
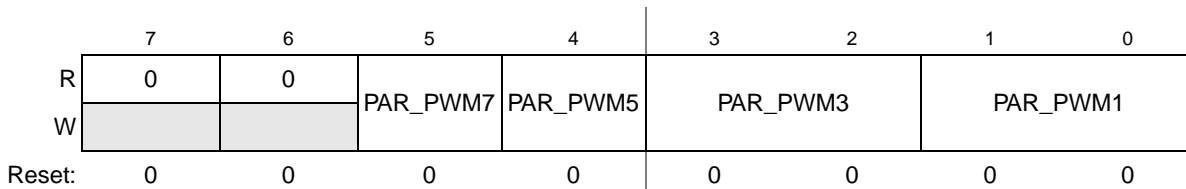
Figure 13-44. PWM Pin Assignment (PAR_PWM)

Table 13-18. PAR_PWM Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 PAR_PWM7	PWM7 pin assignment. 0 PWM7 pin configured as GPIO. 1 PWM7 pin configured for PWM 7 function.

Table 13-18. PAR_PWM Field Descriptions (continued)

Field	Description
4 PAW_PWM5	PWM5 pin assignment. 0 PWM5 pin configured as GPIO. 1 PWM5 pin configured for PWM5 function.
3–2 PAR_PWM3 1–0 PAR_PWM1	PWM pin assignment. These bit fields configure the PWM output pins for one of their primary functions or GPIO. |	PAR_PWM3	PAR_PWM1
---	---	---
00	GPIO	GPIO
01	DT3IN	DT2IN
10	DT3OUT	DT2OUT
11	PWM3	PWM1	

13.3.5.12 LCD Controller DataFEC Pin Assignment Register (PAR_LCDDATA)

The PAR_LCDDATA register controls the functions of the LCDC data pins.

Address: 0xFC0A_405D (PAR_LCDDATA)   Access: User read/write

	7	6	5	4	3	2	1	0
R W	PAR_LD17		PAR_LD16		PAR_LD15_8		PAR_LD7_0	
Reset:	0	0	0	0	0	0	0	0
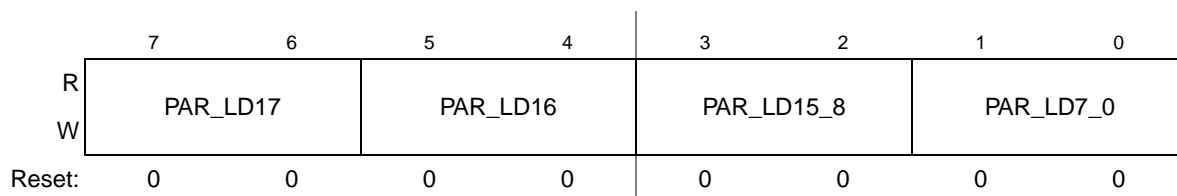
Figure 13-45. LCD Data Pin Assignment (PAR_LCDDATA)

Table 13-19. PAR_LCDDATA Field Descriptions

Field	Description
7–6 PAR_LD17 5–4 PAR_LD16 3–2 PAR_LD15_8 1–0 PAR_LD7_0	LCD data pin assignment. These bit fields configure the LCD data pins for one of their primary functions or GPIO. |	PAR_LD17	PAR_LD16	PAR_LD15_8	PAR_LD7_0
---	---	---	---	---
00	GPIO	GPIO	GPIO	GPIO
01	Reserved	Reserved	Reserved	Reserved
10	CANTX	CANRX	Reserved	Reserved
11	LCD_D17	LCD_D16	LCD_D[15:8]	LCD_D[7:0]	

13.3.5.13 LCD Controller Control Pin Assignment Register (PAR_LCDCTL)

The PAR_LCDCTL register controls the functions of the LCDC control pins.

Address: 0xFC0A_405E (PAR_LCDCTL)   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	0	PAR_ ACD_ OE	PAR_ FLM_ VSYNC	PAR_ LP_ HSYNC	PAR_ LSCLK	PAR_ CON TRAST	PAR_ SPL_ SPR	PAR_ REV	PAR_ PS	PAR_ CLS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-46. LCD Control Pin Assignment (PAR_LCDCTL)

Table 13-20. PAR_LCDCTL Field Descriptions

Field	Description
15–9	Reserved, should be cleared.
8 PAR_ACD_OE	LCD_ACD/OE pin assignment. 0 LCD_ACD/OE pin configured as GPIO. 1 LCD_ACD/OE pin configured for LCD controller ACD/OE function.
7 PAR_FLM_VSYNC	LCD_FLM/VSYNC pin assignment. 0 LCD_FLM/VSYNC pin configured as GPIO. 1 LCD_FLM/VSYNC pin configured for LCD controller FLM/VSYNC function.
6 PAR_LP_HSYNC	LCD_LP/HSYNC pin assignment. 0 LCD_LP/HSYNC pin configured as GPIO. 1 LCD_LP/HSYNC pin configured for LCD controller LP/HSYNC function.
5 PAR_LSCLK	LCD_LSCLK pin assignment. 0 LCD_LSCLK pin configured as GPIO. 1 LCD_LSCLK pin configured for LCD controller LSCLK function.
4 PAR_CONTRAST	LCD_CONTRAST pin assignment. 0 LCD_CONTRAST pin configured as GPIO. 1 LCD_CONTRAST pin configured for LCD controller CONTRAST function.
3 PAR_SPL_SPR	LCD_SPL_SPR pin assignment. 0 LCD_SPL_SPR pin configured as GPIO. 1 LCD_SPL_SPR pin configured for LCD controller ACD/OE function.
2 PAR_REV	LCD_REV pin assignment. 0 LCD_REV pin configured as GPIO. 1 LCD_REV pin configured for LCD controller REV function.
1 PAR_PS	LCD_PS pin assignment. 0 LCD_PS pin configured as GPIO. 1 LCD_PS pin configured for LCD controller PS function.
0 PAR_CLS	LCD_CLS pin assignment. 0 LCD_CLS pin configured as GPIO. 1 LCD_CLS pin configured for LCD controller CLS function.

13.3.6 FlexBus Mode Select Control Register (MSCR_FLEXBUS)

The MSCR_FLEXBUS register controls the output mode selects of the following FlexBus pins: FB_A[23:0], D[31:0], $\overline{BE}/\overline{BWE}$[3:0], $\overline{OE}$, R/$\overline{W}$, $\overline{FB\_CS}$[5:0], $\overline{TA}$ and $\overline{TS}$.

Address: 0xFC0A_4064 (MSCR_FLEXBUS)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	MSCR_DUPPER		MSCR_DLOWER		MSCR_ADDRCTL	
W								
Reset:	0	0	1	1	1	1	1	1
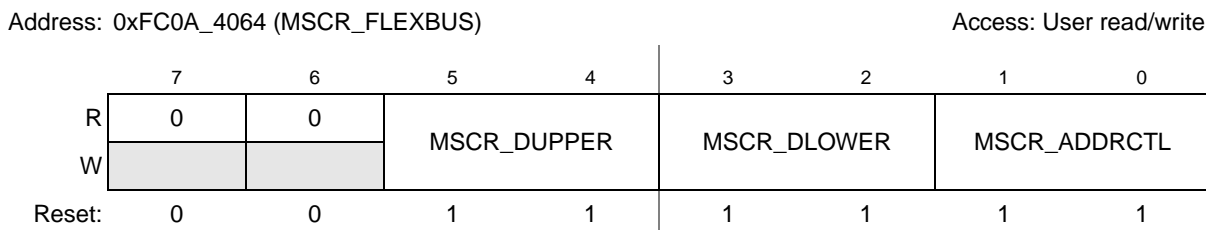
Figure 13-47. FlexBus Mode Select Control Register (MSCR_FLEXBUS)

Table 13-21. MSCR_FLEXBUS Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5–4 MSCR_ DUPPER	FB_D[31:16] mode select control. These bit fields control the strength of the FlexBus upper data pins. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
3–2 MSCR_ DLOWER	FB_D[15:0] mode select control. These bit fields control the strength of the FlexBus lower data pins. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
1–0 MSCR_ ADDRCTL	FB_A[23:0], $\overline{BE}/\overline{BWE}$[3:0], $\overline{OE}$, R/$\overline{W}$, $\overline{FB\_CS}$[5:0], $\overline{TA}$, and $\overline{TS}$ mode select control. These bit fields control the strength of the FlexBus address and control pins. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.

13.3.7 SDRAM Mode Select Control Register (MSCR_SDRAM)

The MSCR_SDRAM register controls the output mode selects of the following dedicated SDRAM pins: SD_A10, $\overline{SD\_CAS}$, SD_CKE, SD_CLK, $\overline{SD\_CLK}$, $\overline{SD\_CS0}$, SD_DQS[3:2], $\overline{SD\_RAS}$, SD_SDRDQS, and $\overline{SD\_WE}$.

Address: 0xFC0A_4065 (MSCR_SDRAM)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	MSCR_SDCLKB		MSCR_SDCLK		MSCR_SDRAM	
W								
Reset:	0	0	1	1	1	1	1	1
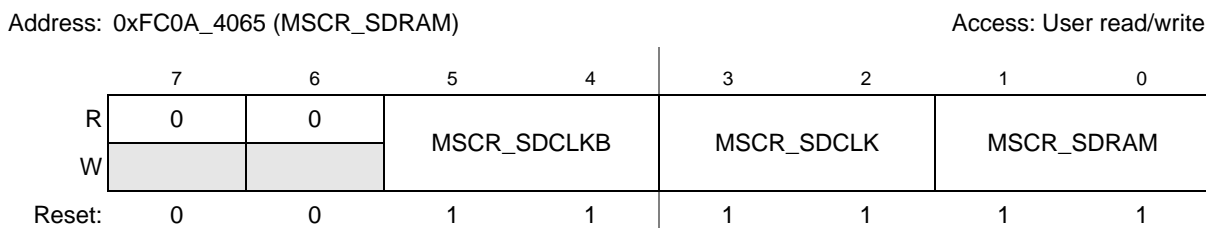
Figure 13-48. SDRAM Mode Select Control Register (MSCR_SDRAM)

Table 13-22. MSCR_SDRAM Field Descriptions

Field	Description
7–6	Reserved, should be cleared
5–4 MSCR_ SDCLKB	$\overline{\text{SD\_CLK}}$ mode select control. These bit fields control the strength of the FlexBus upper data pins. 00 Half strength 1.8V Mobile DDR. 01 Open drain. 10 Full strength 1.8V Mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
3–2 MSCR_ SDCLK	SD_CLK mode select control. These bit fields control the strength of the FlexBus lower data pins. 00 Half strength 1.8V Mobile DDR. 01 Open drain. 10 Full strength 1.8V Mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
1–0 MSCR_ SDRAM	SD_A10, $\overline{\text{SD\_CAS}}$, SD_CKE, $\overline{\text{SD\_CS0}}$, SD_DQS[3:2], $\overline{\text{SD\_RAS}}$, SD_SDRDQS, SD_WE mode select control. These bit fields control the strength of the FlexBus address and control pins. 00 Half strength 1.8V Mobile DDR. 01 Open drain. 10 Full strength 1.8V Mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.

13.3.8 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.

Register (DSCR_*x*)	Pins Affected
DSCR_I2C	I2C_SDA and I2C_SCL
DSCR_PWM	PWM[7,5,3,1]
DSCR_FEC	FEC_MDC and FEC_MDIO
DSCR_QSPI	QSPI_PCS[2:0], QSPI_SCK, QSPI_DIN, and QSPI_DOUT
DSCR_TIMER	DT3IN, DT2IN, DT1IN, and DT0IN
DSCR_SSI	SSI_MCLK, SSI_BCLK, SSI_RXD, and SSI_TXD
DSCR_LCD	LCD_D[17:0], ACD/OE, CLS, CONTRAST, FLM/SYNC, LP/VSYNC, LSCLK, PS, REV, and SPL_SPR.
DSCR_DEBUG	PSTCLK, PST[3:0], DDATA[3:0], ALLPST, and DSO
DSCR_IRQ	IRQ[7:1]

Address: 0xFC0A_4068 (DSCR_I2C)   Access: User read/write
0xFC0A_4069 (DSCR_PWM)
0xFC0A_406A (DSCR_FEC)
0xFC0A_406C (DSCR_QSPI)
0xFC0A_406D (DSCR_TIMER)
0xFC0A_406E (DSCR_SSI)
0xFC0A_406F (DSCR_LCD)
0xFC0A_4070 (DSCR_DEBUG)
0xFC0A_4072 (DSCR_IRQ)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	x_DSE	
W								
Reset:	0	0	0	0	0	0	See Note	1
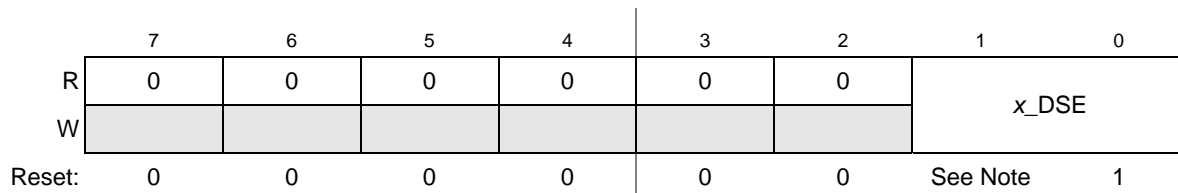
Note: Reset state is 0 when $\overline{RCON}$ = 1, and is value of D[5] when $\overline{RCON}$ = 0.

Figure 13-49. Drive Strength Control Registers (DSCR_*x*)

Table 13-23. DSCR_*x* Field Descriptions

Field	Description
7–2	Reserved, should be cleared
1–0 *x*_DSE	Drive strength control. This bit field controls the drive strength of the various pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.3.8.1   UART Drive Strength Control Register (DSCR_UART)

The DSCR_UART register controls the output drive strengths of the UART1, UART0, and IRQ pins.

Address: 0xFC0A_406B (DSCR_UART)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	UART1_DSE		UART0_DSE	
W								
Reset:	0	0	0	0	See Note	1	See Note	1
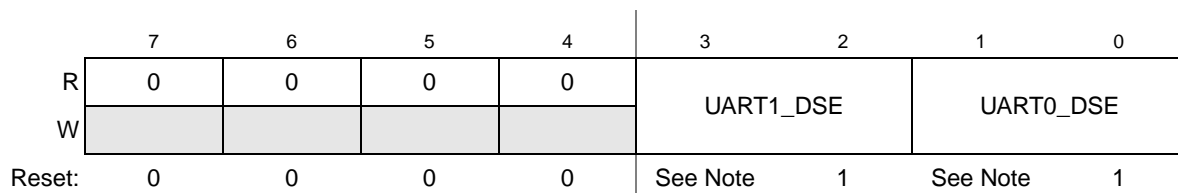
Note: Reset state is 0 when $\overline{RCON}$ = 1, and is value of D[5] when $\overline{RCON}$ = 0.

Figure 13-50. UART Drive Strength Control Register (DSCR_UART)

Table 13-24. DSCR_UART Field Descriptions

Field	Description
7–4	Reserved, should be cleared
3–2 UART1_DSE	UART1 drive strength control. This bit field controls the drive strength of the U1RXD, U1TXD, $\overline{\text{U1CTS}}$, and $\overline{\text{U1RTS}}$ pins. 00 10pF 01 20pF 10 30pF 11 50pF
1–0 UART0_DSE	UART0 drive strength control. This bit field controls the drive strength of the U0RXD, U0TXD, $\overline{\text{U0CTS}}$, and $\overline{\text{U0RTS}}$ pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.3.8.2 Clock/Reset Pins Drive Strength Control Register (DSCR_CLKRST)

The DSCR_CLKRST register controls the output drive strengths of the $\overline{\text{RSTOUT}}$ pin and mode of the FB_CLK pin.

Address: 0xFC0A_4071 (DSCR_CLKRST)   Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	RSTOUT_DSE		MSCR_FBCLK	
W								
Reset:	0	0	0	0	See Note	1	1	1

Note: Reset state is 0 when $\overline{\text{RCON}}$ = 1, and is value of D[5] when $\overline{\text{RCON}}$ = 0.

Figure 13-51. Clock/Reset Drive Strength Control Register (DSCR_CLKRST)

Table 13-25. DSCR_CLKRST Field Descriptions

Field	Description
7–4	Reserved, should be cleared
3–2 RSTOUT_DSE	$\overline{\text{RSTOUT}}$ drive strength control. This bit field controls the drive strength of the $\overline{\text{RSTOUT}}$ pin. 00 10pF 01 20pF 10 30pF 11 50pF
1–0 MSCR_FBCLK	FB_CLK mode select control. This bit field controls the strength of the FlexBus clock pin. 00 Half strength 1.8V. 01 Open drain. 10 Full strength 1.8V. 11 2.5V or 3.3V with roughly equal rise and fall delays.

13.4 Functional Description

13.4.1 Overview

Initial pin function is determined during reset configuration. The pin assignment registers allow the user 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.

13.4.2 Port Digital I/O Timing

Input data on all pins configured as general purpose input is synchronized to the rising edge of the bus clock, FB_CLK, as shown in Figure 13-52.
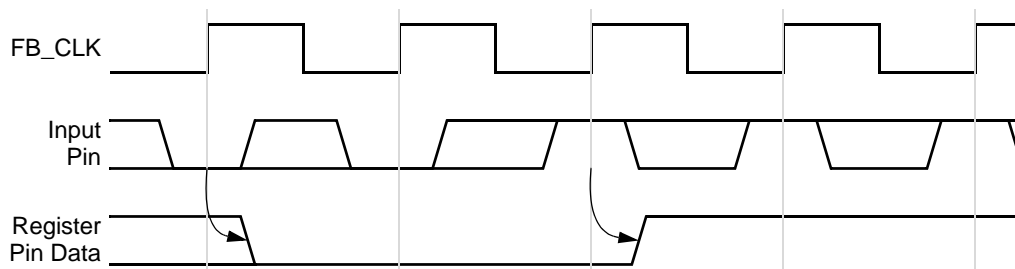
Figure 13-52. General Purpose Input Timing

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 13-53.
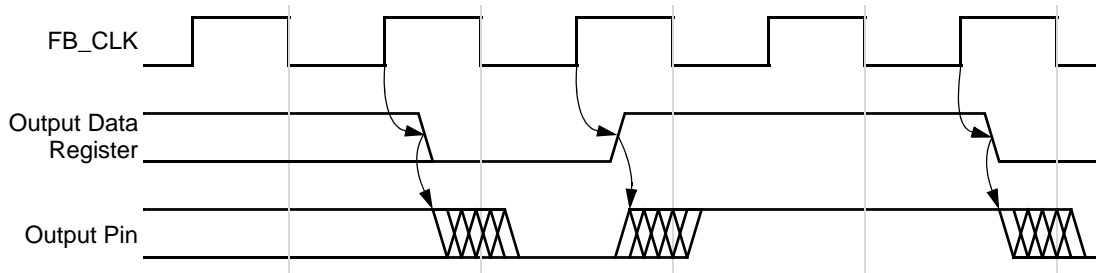
Figure 13-53. General Purpose Output Timing

13.5 Initialization/Application Information

The initialization for the ports module is done during reset configuration. All registers are reset to a predetermined state. Refer to Section 13.3, "Memory Map/Register Definition," for more details on reset and initialization.

Chapter 14
Interrupt Controller Modules

14.1   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 (ICR0n, ICR1n) 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.

14.1.1   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 14.3.1.3, "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.

14.2 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 14-2. The base addresses for the interrupt controllers are listed below.

Table 14-1. 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 14.2.10, "Software and Level 1 – 7 IACK Registers (SWIACKn, L1IACKn – L7IACKn)" for more information

Table 14-2. 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	14.2.1/14-4
0xFC04_8004	Interrupt Pending Register Low (IPRL0)	32	R	0x0000_0000	14.2.1/14-4
0xFC04_8008	Interrupt Mask Register High (IMRH0)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_800C	Interrupt Mask Register Low (IMRL0)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_8010	Interrupt Force Register High (INTFRCH0)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_8014	Interrupt Force Register Low (INTFRCL0)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_801A	Interrupt Configuration Register (ICONFIG)	16	R/W	0x0000	14.2.4/14-7
0xFC04_801C	Set Interrupt Mask (SIMR0)	8	W	0x00	14.2.5/14-8
0xFC04_801D	Clear Interrupt Mask (CIMR0)	8	W	0x00	14.2.6/14-9
0xFC04_801E	Current Level Mask (CLMASK)	8	R/W	0x0F	14.2.7/14-9
0xFC04_801F	Saved Level Mask (SLMASK)	8	R/W	0x0F	14.2.8/14-10
0xFC04_8040 + n (n=0:63)	Interrupt Control Registers (ICR0n)	8	R/W	0x00	14.2.9/14-11
0xFC04_80E0	Software Interrupt Acknowledge (SWIACK0)	8	R	0x00	14.2.10/14-15
0xFC04_80E0 + 4n (n=1:7)	Level n Interrupt Acknowledge Registers (LnIACK0)	8	R	0x18	14.2.10/14-15
Interrupt Controller 1					
0xFC04_C000	Interrupt Pending Register High (IPRH1)	32	R	0x0000_0000	14.2.1/14-4
0xFC04_C004	Interrupt Pending Register Low (IPRL1)	32	R	0x0000_0000	14.2.1/14-4
0xFC04_C008	Interrupt Mask Register High (IMRH1)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_C00C	Interrupt Mask Register Low (IMRL1)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_C010	Interrupt Force Register High (INTFRCH1)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_C014	Interrupt Force Register Low (INTFRCL1)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_C01C	Set Interrupt Mask (SIMR1)	8	W	0x00	14.2.5/14-8
0xFC04_C01D	Clear Interrupt Mask (CIMR1)	8	W	0x00	14.2.5/14-8
0xFC04_C040 + n (n=1:63)	Interrupt Control Registers (ICR1n)	8	R/W	0x00	14.2.9/14-11
0xFC04_C0E0	Software Interrupt Acknowledge (SWIACK1)	8	R	0x00	14.2.10/14-15
0xFC04_C0E0 + 4n (n=1:7)	Level n Interrupt Acknowledge Registers (LnIACK1)	8	R	0x18	14.2.10/14-15
Global IACK Registers					

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC05_40E0	Global Software Interrupt Acknowledge (GSWIACK)	8	R	0x00	14.2.10/14-15
0xFC05_40E0 + 4*n* (*n*=1:7)	Global Level *n* Interrupt Acknowledge Registers (GL*n*IACK)	8	R	0x18	14.2.10/14-15

14.2.1 Interrupt Pending Registers (IPRH*n*, IPRL*n*)

The IPRH*n* and IPRL*n* registers, Figure 14-1 and Figure 14-2, 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)   Access: User read-only
0xFC04_C000 (IPRH1)
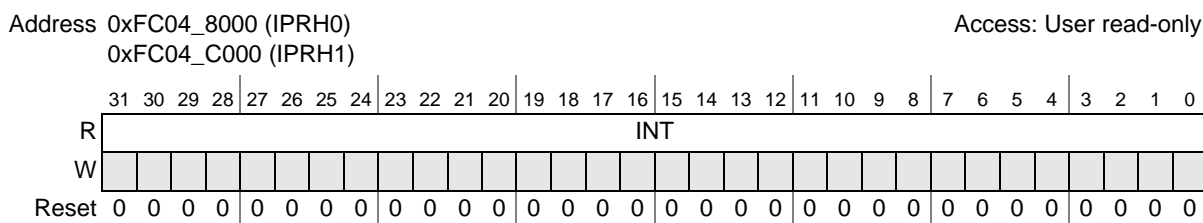
Figure 14-1. Interrupt Pending Register High (IPRH*n*)

Table 14-3. 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)   Access: User read-only
0xFC04_C004 (IPRL1)
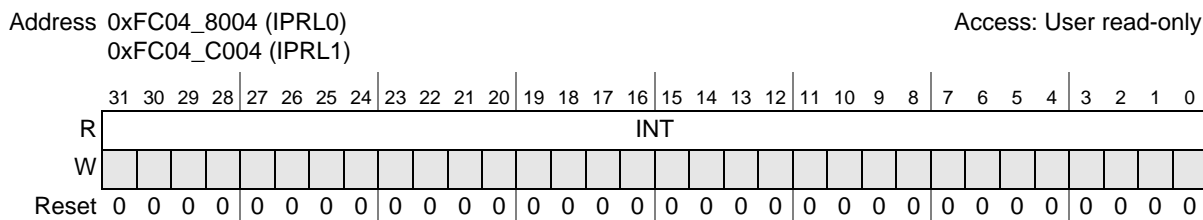
Figure 14-2. Interrupt Pending Register Low (IPRL*n*)

Table 14-4. 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

14.2.2 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)   Access: User read/write
         0xFC04_C008 (IMRH1)

	31	30	29	28	27	26	25	24	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_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
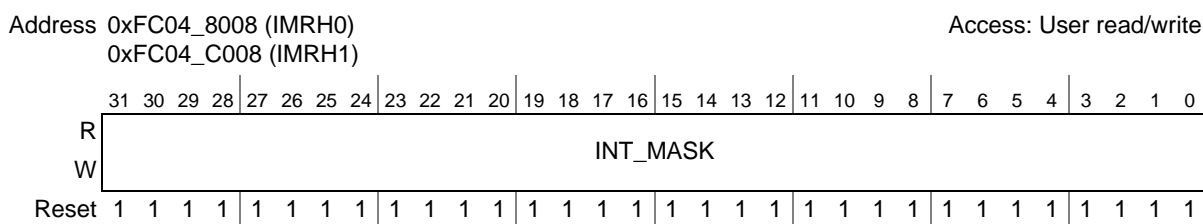
Figure 14-3. Interrupt Mask Register High (IMRH*n*)

Table 14-5. 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)   Access: User read/write
         0xFC04_C00C (IMRL1)

	31	30	29	28	27	26	25	24	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_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 14-4. Interrupt Mask Register Low (IMRL*n***)**

Table 14-6. 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

14.2.3 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)   Access: User read/write
         0xFC04_C010 (INTFRCH1)

	31	30	29	28	27	26	25	24	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																INTFRCH																
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
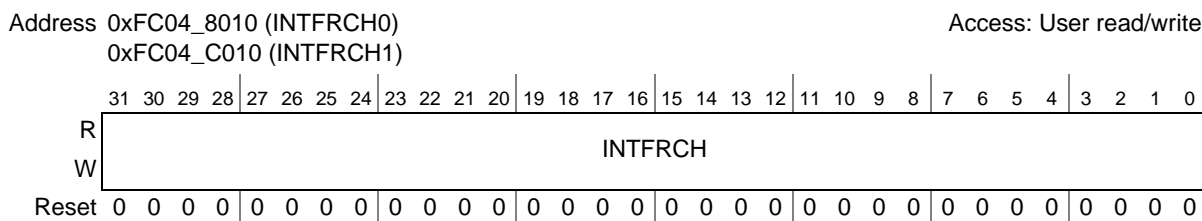
Figure 14-5. Interrupt Force Register High (INTFRCH*n***)**

Table 14-7. 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)   Access: User read/write
              0xFC04_C014 (INTFRCL1)

	31	30	29	28	27	26	25	24	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
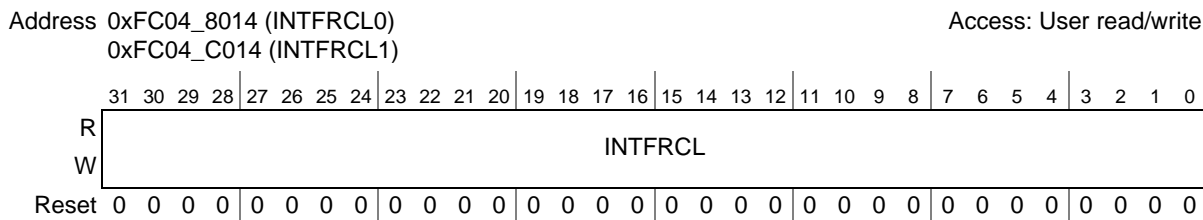
Figure 14-6. Interrupt Force Register Low (INTFRCL*n*)

Table 14-8. 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

14.2.4 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
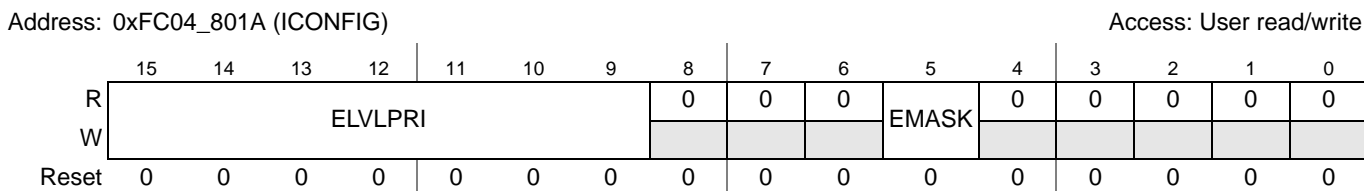
Figure 14-7. Interrupt Configuration Register (ICONFIG)

Table 14-9. 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.

14.2.5 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)   Access: User write-only
         0xFC04_C01C (SIMR1)

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SALL	SIMR					
Reset:	0	0	0	0	0	0	0	0
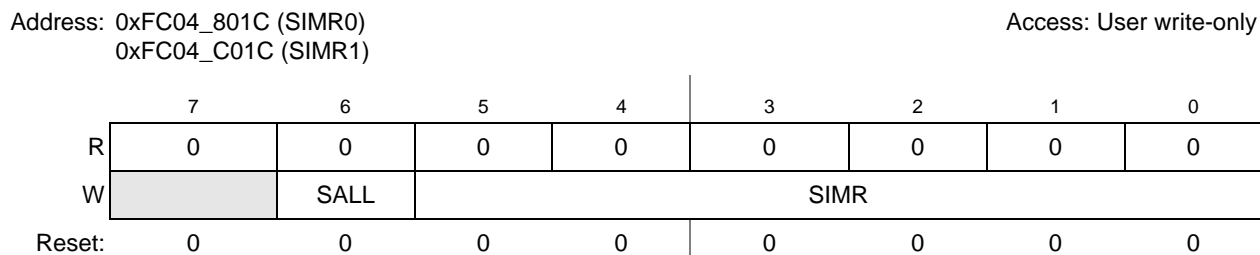
Figure 14-8. Set Interrupt Mask Register (SIMR*n*)

Table 14-10. 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.

14.2.6 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
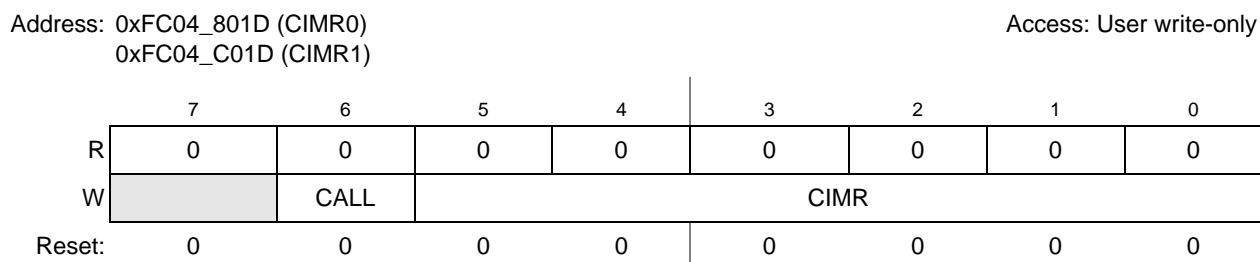
Figure 14-9. Clear Interrupt Mask Register (CIMR*n*)

Table 14-11. 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.

14.2.7 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
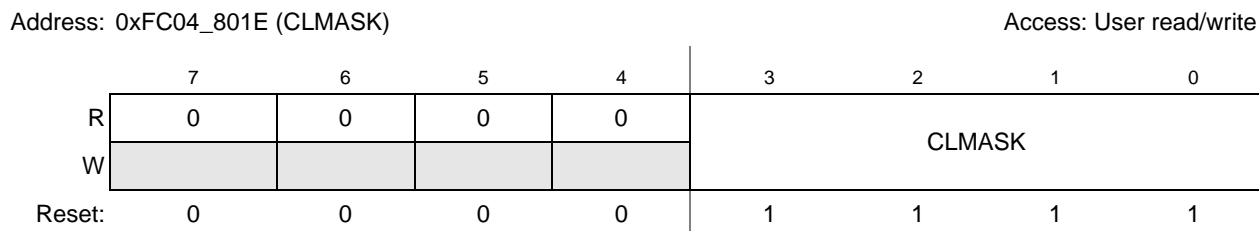
Figure 14-10. Current Level Mask Register (CLMASK)

Table 14-12. 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.

14.2.8   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
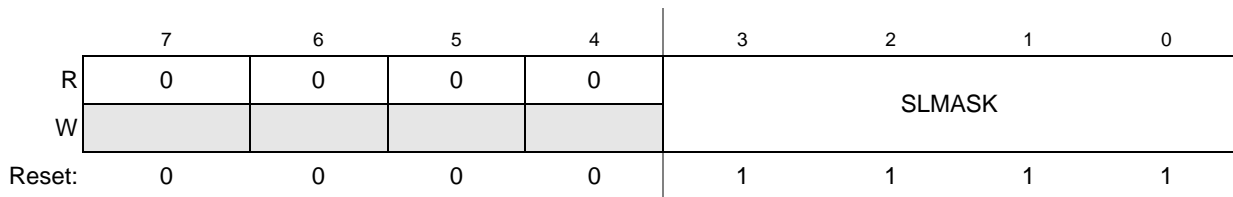
Figure 14-11. Saved Level Mask Register (SLMASK)

Table 14-13. 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.

14.2.9 Interrupt Control Register (ICR0n, ICR1n, (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)   Access: User read/write
         0xFC04_C040 – 7F (ICR100 – ICR163)

	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
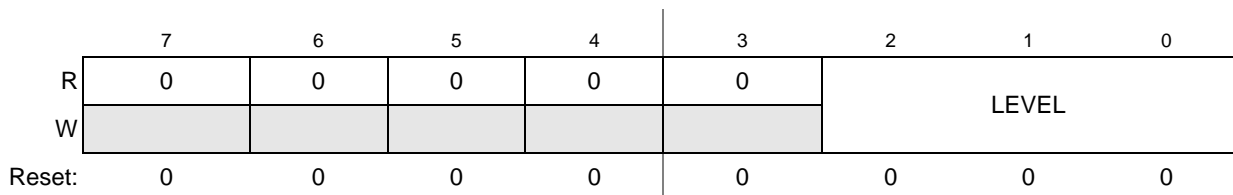
Figure 14-12. Interrupt Control Registers (ICR0n, ICR1n)

Table 14-14. ICRn 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.

14.2.9.1　Interrupt Sources

Table 14-15 and Table 14-16 list the interrupt sources for each interrupt request line for INTC0 and INTC1.

Table 14-15. 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[ERRn]	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 14-15. Interrupt Source Assignment For INTC0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
30	I²C	I2SR[IIF]	I²C Interrupt	Write I2SR[IIF] = 0
31	QSPI	QIR register	QSPI interrupt	Write 1 to appropriate QIR 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	FEC	EIR[TXF]	Transmit frame interrupt	Write EIR[TXF] = 1
37		EIR[TXB]	Transmit buffer interrupt	Write EIR[TXB] = 1
38		EIR[UN]	Transmit FIFO underrun	Write EIR[UN] = 1
39		EIR[RL]	Collision retry limit	Write EIR[RL] = 1
40		EIR[RXF]	Receive frame interrupt	Write EIR[RXF] = 1
41		EIR[RXB]	Receive buffer interrupt	Write EIR[RXB] = 1
42		EIR[MII]	MII interrupt	Write EIR[MII] = 1
43		EIR[LC]	Late collision	Write EIR[LC] = 1
44		EIR[HBERR]	Heartbeat error	Write EIR[HBERR] = 1
45		EIR[GRA]	Graceful stop complete	Write EIR[GRA] = 1
46		EIR[EBERR]	Ethernet bus error	Write EIR[EBERR] = 1
47		EIR[BABT]	Babbling transmit error	Write EIR[BABT] = 1
48		EIR[BABR]	Babbling receive error	Write EIR[BABR] = 1
49–61	Not Used			
62	SCM	SCMIR[CFEI]	Core bus error interrupt	Write SCMIR[CFEI] = 1
63	Not Used			

Table 14-16. Interrupt Source Assignment for INTC1

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0	FLEX CAN		Logical OR of CAN0's MB requests	Write 1 to BUF*n*I after reading as 1
1		BOFF_INT	Bus-Off Interrupt	Write 0 to BOFF_INT
2		Not Used		
3		ERR_INT	Error Interrupt	Read reported error bits in ESR or write 0 to ERR_INT
4		BUF0I	Message Buffer 0 Interrupt	Write 1 to BUF0I after reading as 1
5		BUF1I	Message Buffer 1 Interrupt	Write 1 to BUF1I after reading as 1
6		BUF2I	Message Buffer 2 Interrupt	Write 1 to BUF2I after reading as 1
7		BUF3I	Message Buffer 3 Interrupt	Write 1 to BUF3I after reading as 1
8		BUF4I	Message Buffer 4 Interrupt	Write 1 to BUF4I after reading as 1
9		BUF5I	Message Buffer 5 Interrupt	Write 1 to BUF5I after reading as 1
10		BUF6I	Message Buffer 6 Interrupt	Write 1 to BUF6I after reading as 1
11		BUF7I	Message Buffer 7 Interrupt	Write 1 to BUF7I after reading as 1
12		BUF8I	Message Buffer 8 Interrupt	Write 1 to BUF8I after reading as 1
13		BUF9I	Message Buffer 9 Interrupt	Write 1 to BUF9I after reading as 1
14		BUF10I	Message Buffer 10 Interrupt	Write 1 to BUF10I after reading as 1
15		BUF11I	Message Buffer 11 Interrupt	Write 1 to BUF11I after reading as 1
16		BUF12I	Message Buffer 12 Interrupt	Write 1 to BUF12I after reading as 1
17		BUF13I	Message Buffer 13 Interrupt	Write 1 to BUF13I after reading as 1
18		BUF14I	Message Buffer 14 Interrupt	Write 1 to BUF14I after reading as 1
19		BUF15I	Message Buffer 15 Interrupt	Write 1 to BUF15I after reading as 1
20–39		Not Used		
40	RNG	EI	RNG interrupt flag	Write RNGCR[CI] = 1
41	SKHA	INT	SKHA interrupt flag	Write SKCMR[CI] = 1
42	MDHA	MI	MDHA interrupt flag	Write MDCMR[CI] = 1
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	USB Host	USB_STS	USB host interrupt	Write 1 to corresponding bit in the USB_STS.
49	SSI	SSI_ISR	SSI interrupt	Various, see chapter for details.
50	PWM	PWMSDN[IF]	PWM interrupt	Write PWMSDN[IF] = 1

Table 14-16. Interrupt Source Assignment for INTC1 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
51	LCDC	LCD_ISR	LCD controller interrupt	Read the LCD_ISR register.
52	RTC	RTC_ISR	Real time clock interrupt	Write one to corresponding bit in RTC_ISR.
53	CCM	UHCSR or UOCSR	USB status Interrupt	Read UHCSR or read UOCSR.
54–63	Not Used			

14.2.10 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)   Access: User read-only
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

	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
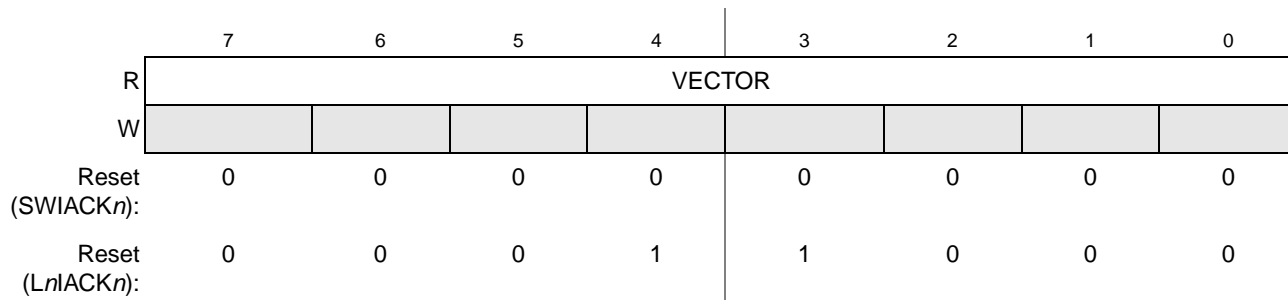
Figure 14-13. Software and Level *n* IACK Registers (SWIACK*n*, L1IACK*n* – L7IACK*n*)

Table 14-17. 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.

14.3 Functional Description

14.3.1 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 14.2.9.1, "Interrupt Sources"). The higher numbered interrupt source has priority over the lower numbered interrupt source. See the below table for an example.

Table 14-18. 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

14.3.1.1 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.

14.3.1.2 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.

14.3.1.3 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.

14.3.2   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 ise serviced first. If INTC1 has an active interrupt with a higher level than the highest INTC0 interrupt, the INTC1 interrupt is serviced first.

14.3.3   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 8, "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 8.2.1, "Wake-up Control Register".

14.4 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.

14.4.1 Interrupt Service Routines

This section focuses on the interaction of the interrupt masking functionality with the service routine. Figure 14-14 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.
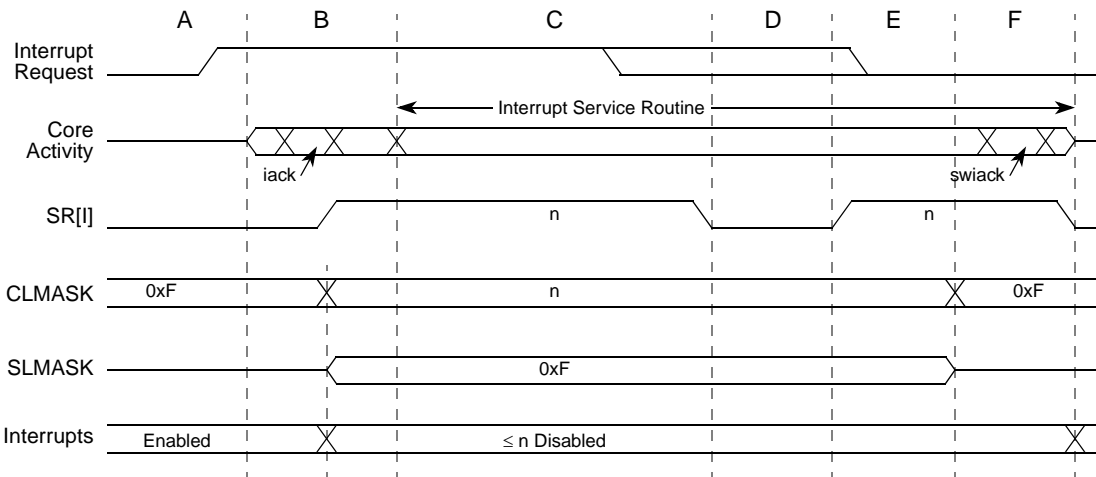
Figure 14-14. Interrupt Service Routine and Masking

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 15
Edge Port Module (EPORT)

15.1 Introduction

The edge port module (EPORT) has eight interrupt pins, $\overline{IRQ7}$ – $\overline{IRQ0}$. 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.
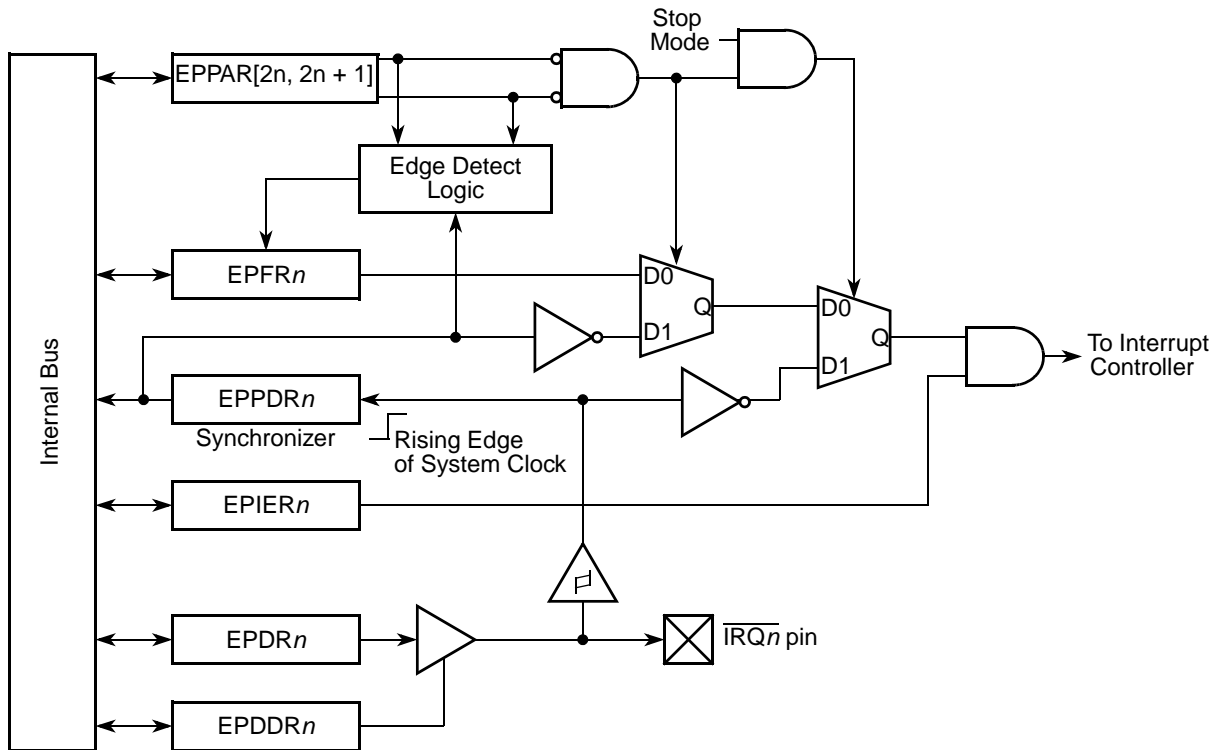
Figure 15-1. EPORT Block Diagram

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 13, "General Purpose I/O Module") prior to configuring the edge-port module.

15.2 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 8, "Power Management". Table 15-1 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 15-1. 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.

15.3 Interrupt/GPIO Pin 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.

15.4 Memory Map/Register Definition

This subsection describes the memory map and register structure. Refer to Table 15-2 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 15-2. 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	15.4.1/15-3
0xFC09_4002	EPORT Data Direction Register (EPDDR)	8	R/W	0x00	15.4.2/15-4
0xFC09_4003	EPORT Interrupt Enable Register (EPIER)	8	R/W	0x00	15.4.3/15-5
Supervisor/User Access Registers					
0xFC09_4004	EPORT Data Register (EPDR)	8	R/W	0xFF	15.4.4/15-5
0xFC09_4005	EPORT Pin Data Register (EPPDR)	8	R	See Section	15.4.5/15-5
0xFC09_4006	EPORT Flag Register (EPFR)	8	R/W	0x00	15.4.6/15-6

[1] User access to supervisor-only address locations have no effect and result in a bus error.

15.4.1 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 15-2. EPORT Pin Assignment Register (EPPAR)

Table 15-3. 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 $\overline{\text{IRQ}n}$ 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 $\overline{\text{IRQ}n}$ level-sensitive 01 Pin $\overline{\text{IRQ}n}$ rising edge triggered 10 Pin $\overline{\text{IRQ}n}$ falling edge triggered 11 Pin $\overline{\text{IRQ}n}$ falling edge and rising edge triggered

15.4.2 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
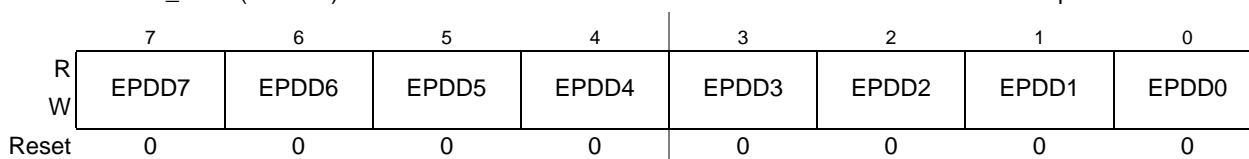
Figure 15-3. EPORT Data Direction Register (EPDDR)

Table 15-4. 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

15.4.3 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
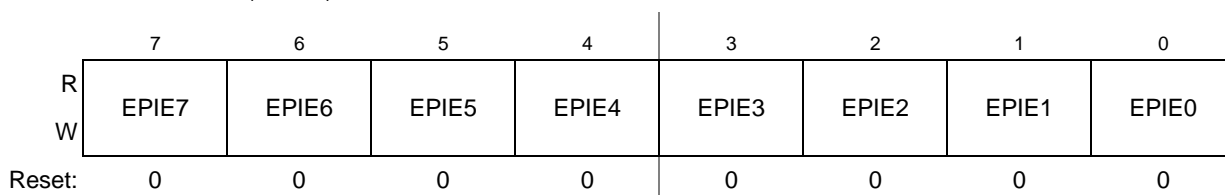
Figure 15-4. EPORT Port Interrupt Enable Register (EPIER)

Table 15-5. 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

15.4.4 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
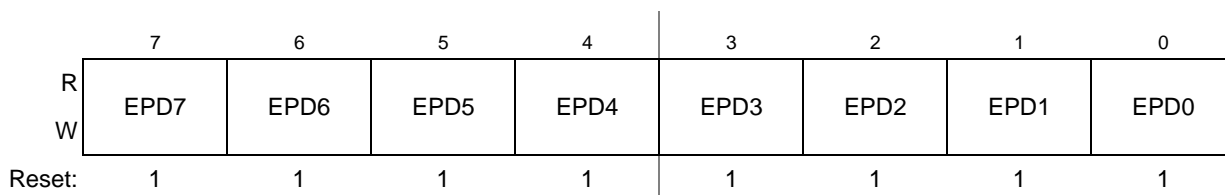
Figure 15-5. EPORT Port Data Register (EPDR)

Table 15-6. 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.

15.4.5 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:	[$\overline{IRQ7}$]	[$\overline{IRQ6}$]	[$\overline{IRQ5}$]	[$\overline{IRQ4}$]	[$\overline{IRQ3}$]	[$\overline{IRQ2}$]	[$\overline{IRQ1}$]	[$\overline{IRQ0}$]
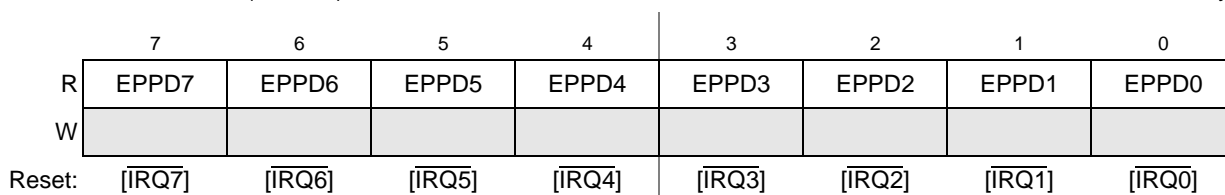
Figure 15-6. EPORT Port Pin Data Register (EPPDR)

Table 15-7. EPPDR Field Descriptions

Field	Description
7–0 EPPDn	Edge port pin data bits. The read-only EPPDR reflects the current state of the EPORT pins $\overline{IRQ7}$ – $\overline{IRQ0}$. Writing to EPPDR has no effect, and the write cycle terminates normally. Reset does not affect EPPDR.

15.4.6 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 W	EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	EPF0
Reset:	0	0	0	0	0	0	0	0
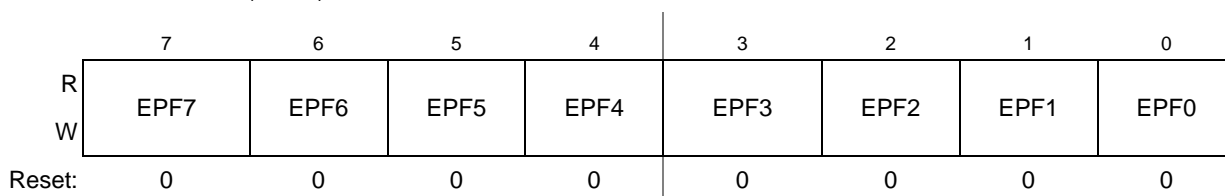
Figure 15-7. EPORT Port Flag Register (EPFR)

Table 15-8. EPFR Field Descriptions

Field	Description
7–0 EPFn	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 (EPPARn = 00), pin transitions do not affect this register. 0 Selected edge for $\overline{IRQ}n$ pin has not been detected. 1 Selected edge for $\overline{IRQ}n$ pin has been detected.

Chapter 16
Enhanced Direct Memory Access (eDMA)

16.1 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.

16.2 Block Diagram

Figure 16-1 is a block diagram of the eDMA module.
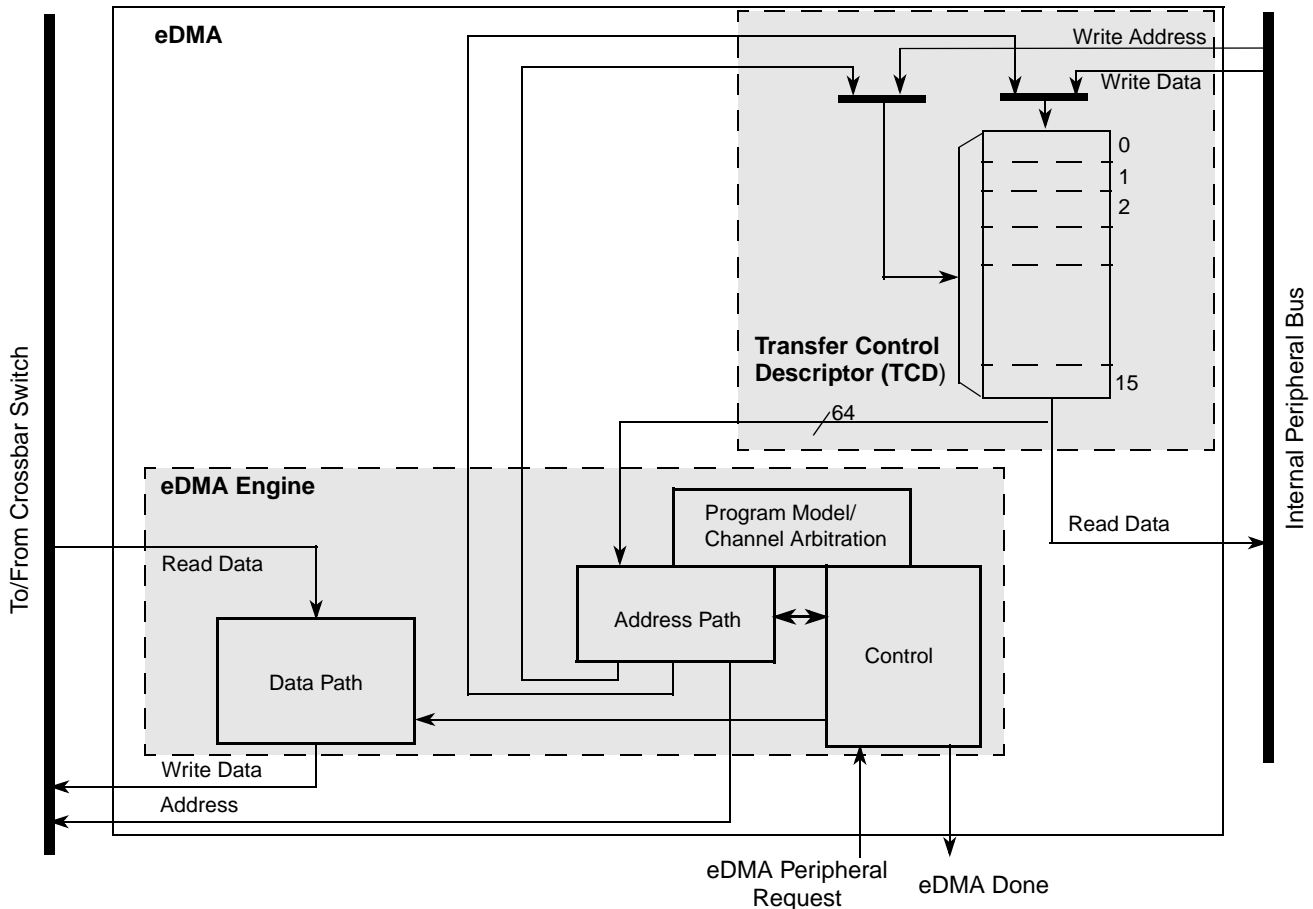
Figure 16-1. eDMA Block Diagram

MCF5329 Reference Manual, Rev 3

16.3   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.

16.4   Modes of Operation

16.4.1   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.

16.4.2 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.

16.5 External Signal Description

This section describes the external signals of the eDMA controller.

Table 16-1. 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.

16.5.1 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 16-2).
- 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.
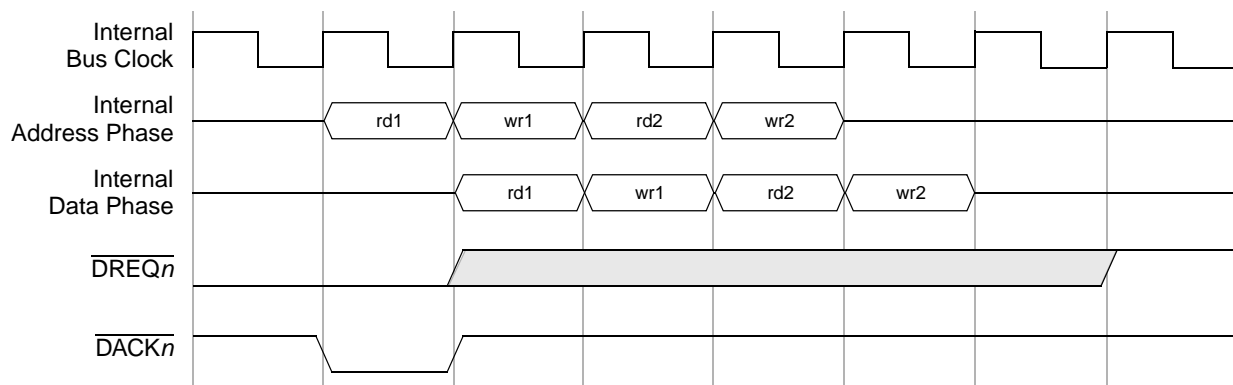
Figure 16-2. $\overline{\text{DREQ}n}$ and $\overline{\text{DACK}n}$ Timing

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.

16.6 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 16-2. 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	16.6.1/16-4
0xFC04_4004	eDMA Error Status Register (EDMA_ES)	32	R	0x0000_0000	16.6.2/16-5
0xFC04_400E	eDMA Enable Request Register (EDMA_ERQ)	16	R/W	0x0000	16.6.3/16-8
0xFC04_4016	eDMA Enable Error Interrupt Register (EDMA_EEI)	16	R/W	0x0000	16.6.4/16-9
0xFC04_4018	eDMA Set Enable Request (EDMA_SERQ)	8	W	0x00	16.6.5/16-9
0xFC04_4019	eDMA Clear Enable Request (EDMA_CERQ)	8	W	0x00	16.6.6/16-10
0xFC04_401A	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	8	W	0x00	16.6.7/16-11
0xFC04_401B	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	8	W	0x00	16.6.8/16-11
0xFC04_401C	eDMA Clear Interrupt Request Register (EDMA_CINT)	8	W	0x00	16.6.9/16-12
0xFC04_401D	eDMA Clear Error Register (EDMA_CERR)	8	W	0x00	16.6.10/16-13
0xFC04_401E	eDMA Set START Bit Register (EDMA_SSRT)	8	W	0x00	16.6.11/16-13
0xFC04_401F	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	8	W	0x00	16.6.12/16-14
0xFC04_4026	eDMA Interrupt Request Register (EDMA_INT)	32	R/W	0x0000	16.6.13/16-15
0xFC04_402E	eDMA Error Register (EDMA_ERR)	32	R/W	0x0000	16.6.14/16-15
0xFC04_4100 + hex(n)	eDMA Channel n Priority Register (DCHPRIn) for n = 0 – 15	8	R/W	See Section	16.6.15/16-16
0xFC04_5000 + hex(32×n)	Transfer Control Descriptor (TCDn) for n = 0 – 15	256	R/W	See Section	16.6.16/16-17

16.6.1 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 16.6.15, "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 16-3. eDMA Control Register (EDMA_CR)

Table 16-3. eDMA_CR Field Descriptions

Field	Description
31–3	Reserved, must 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.

16.6.2   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 16-4. eDMA Error Status Register (EDMA_ES)

Table 16-4. 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 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 TCDn_SADDR field. TCDn_SADDR is inconsistent with TCDn_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 TCDn_SOFF field. TCDn_SOFF is inconsistent with TCDn_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 TCDn_DADDR field. TCDn_DADDR is inconsistent with TCDn_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 TCDn_DOFF field. TCDn_DOFF is inconsistent with TCDn_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 TCDn_NBYTES or TCDn_CITER fields. • TCDn_NBYTES is not a multiple of TCDn_ATTR[SSIZE] and TCDn_ATTR[DSIZE], or • TCDn_CITER[CITER] is equal to zero, or • TCDn_CITER[E_LINK] is not equal to TCDn_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 TCDn_DLAST_SGA field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCDn_CSR[E_SG] is enabled. TCDn_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.

16.6.3 eDMA Enable Request Register (EDMA_ERQ)

The EDMA_ERQ register provides a bit map for the16 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	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
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-5. eDMA Enable Request Register (EDMA_ERQ)

Table 16-5. 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 16-6.

Table 16-6. DMA Request Summary for eDMA

Channel	Source	Description
0	$\overline{DREQ0}$	External DMA request 0
1	$\overline{DREQ1}$	External DMA request 1
2	—	Reserved/Not used
3	UISR0[FFULL/RXRDY]	UART0 Receive
4	UISR0[TXRDY]	UART0 Transmit
5	UISR1[FFULL/RXRDY]	UART1 Receive
6	UISR1[TXRDY]	UART1 Transmit
7	UISR2[FFULL/RXRDY]	UART2 Receive
8	UISR2[TXRDY]	UART2 Transmit
9	DTER0[CAP] or DTER0[REF] / SSISR[RFF0]	Timer 0 / SSI0 Receive[1]
10	DTER1[CAP] or DTER1[REF] / SSISR[RFF1]	Timer 1 / SSI1 Receive[1]

Table 16-6. DMA Request Summary for eDMA (continued)

Channel	Source	Description
11	DTER2[CAP] or DTER2[REF] / SSISR[TFE0]	Timer 2 / SSI0 Transmit[1]
12	DTER3[CAP] or DTER3[REF] / SSISR[TFE1]	Timer 3 / SSI1 Transmit[1]
13	MDHA FIFO & MDCR[DMAL]	MDHA Input FIFO
14	SKHA Output FIFO & SKCR[ODMAL]	SKHA Output FIFO
15	SKHA Input FIFO & SKCR[IDMAL]	SKHA Input FIFO

[1] For information on how to select between SSI and Timer sources, refer to the CCM chapter ."

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.

16.6.4   eDMA Enable Error Interrupt Registers (EDMA_EEI)

The EDMA_EEI register provides a bit map for the16 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
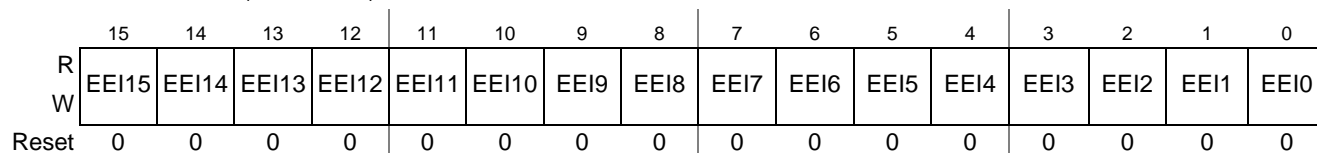
Figure 16-6. eDMA Enable Error Interrupt Register (EDMA_EEI)

Table 16-7. 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.

16.6.5   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 16-7. eDMA Set Enable Request Register (EDMA_SERQ)

Table 16-8. 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

16.6.6   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 16-8. eDMA Clear Enable Request Register (EDMA_CERQ)

Table 16-9. 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.

Table 16-9. EDMA_CERQ Field Descriptions (continued)

Field	Description
5–4	Reserved, must be cleared.
3–0 CERQ	Clear enable request. Clears the corresponding bit in EDMA_ERQ.

16.6.7   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
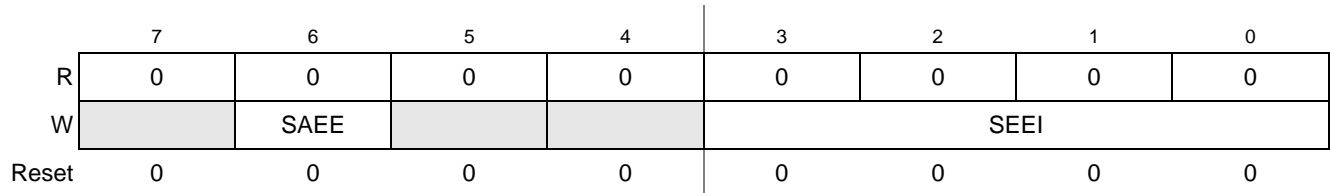
Figure 16-9. eDMA Set Enable Error Interrupt Register (EDMA_SEEI)

Table 16-10. 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.

16.6.8   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		
W		CAEE			CEEI			
Reset	0	0	0	0	0	0	0	0
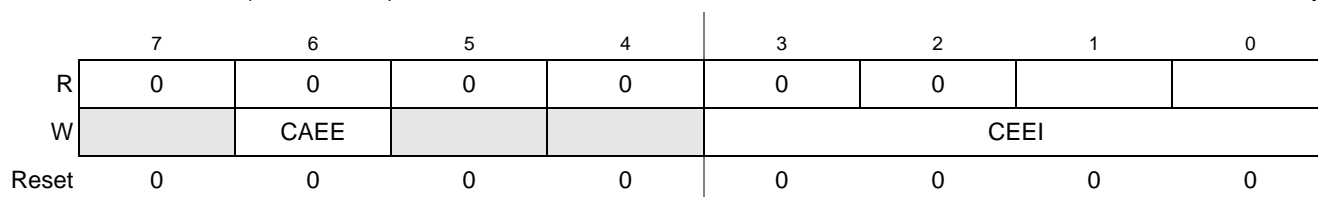
Figure 16-10. eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)

Table 16-11. 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.

16.6.9   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		
W		CAIR			CINT			
Reset	0	0	0	0	0	0	0	0
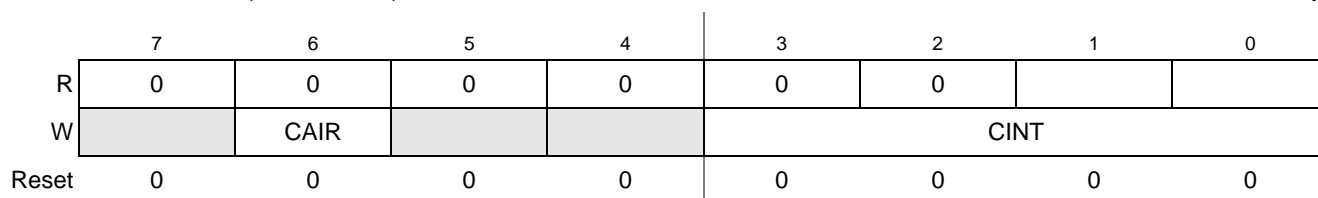
Figure 16-11. eDMA Clear Interrupt Request (EDMA_CINT)

Table 16-12. 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 16-12. 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.

16.6.10   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 16-12. eDMA Clear Error Register (EDMA_CERR)

Table 16-13. 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.

16.6.11   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 16-13. eDMA Set START Bit Register (EDMA_SSRT)

Table 16-14. 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].

16.6.12 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		
W		CADN			CDNE			
Reset	0	0	0	0	0	0	0	0

Figure 16-14. eDMA Clear DONE Status Bit Register (EDMA_CDNE)

Table 16-15. 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].

16.6.13   eDMA Interrupt Request Register (EDMA_INT)

The EDMA_INT provide a bit map for the16 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 16-15. eDMA Interrupt Request Register (EDMA_INT)

Table 16-16. 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.

16.6.14   eDMA Error Register (EDMA_ERR)

The EDMA_ERR provide a bit map for the16 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.

MCF5329 Reference Manual, Rev 3

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 16-16. eDMA Error (EDMA_ERR) Register

Table 16-17. 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.

16.6.15 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	ECP				CHPRI			
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 16-17. eDMA Channel *n* Priority Register (DCHPRI*n*)

Table 16-18. 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.

16.6.16 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 16-19 is a register list of the basic TCD structure.

Table 16-19. 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	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
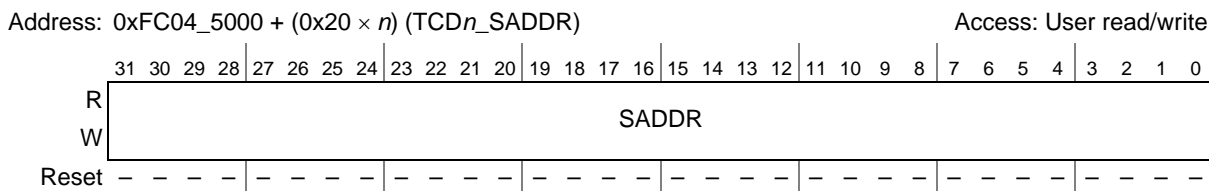
Figure 16-18. TCD*n* Source Address (TCD*n*_SADDR)

Table 16-20. 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	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
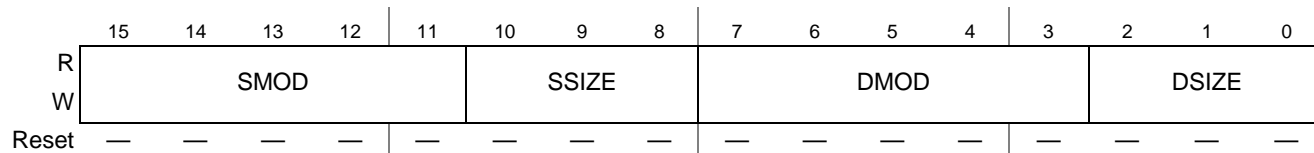
Figure 16-19. TCD*n* Transfer Attributes (TCD*n*_ATTR)

Table 16-21. 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	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
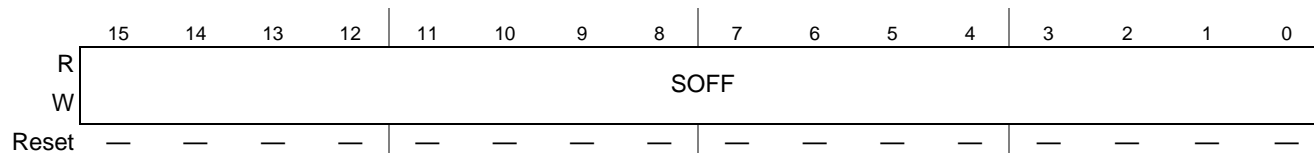
Figure 16-20. TCDn Signed Source Address Offset (TCD*n*_SOFF)

Table 16-22. 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

	31 30 29 28	27 26 25 24	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					NBYTES			
W								
Reset	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
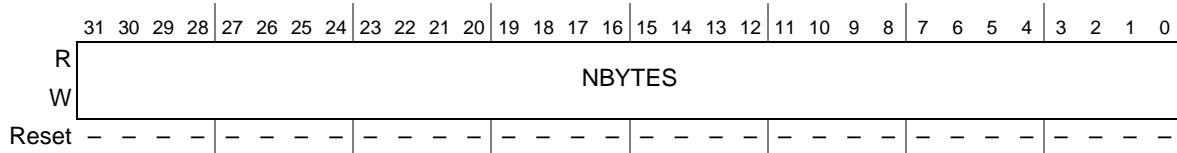
Figure 16-21. TCD*n* Minor Byte Count (TCD*n*_NBYTES)

Table 16-23. 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

	31 30 29 28	27 26 25 24	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					SLAST			
W								
Reset	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –

Figure 16-22. TCD*n* Source Last Address Adjustment (TCD*n*_SLAST)

Table 16-24. 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

	31 30 29 28	27 26 25 24	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					DADDR			
W								
Reset	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
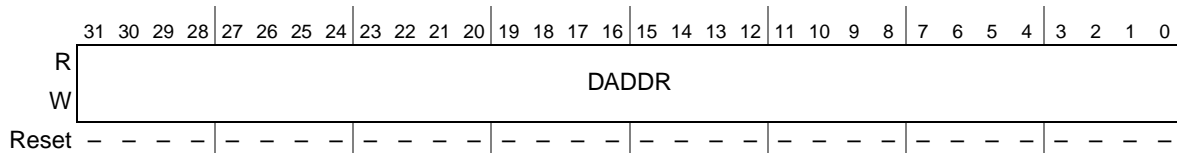
Figure 16-23. TCD*n* Destination Address (TCD*n*_DADDR)

Table 16-25. 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		—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
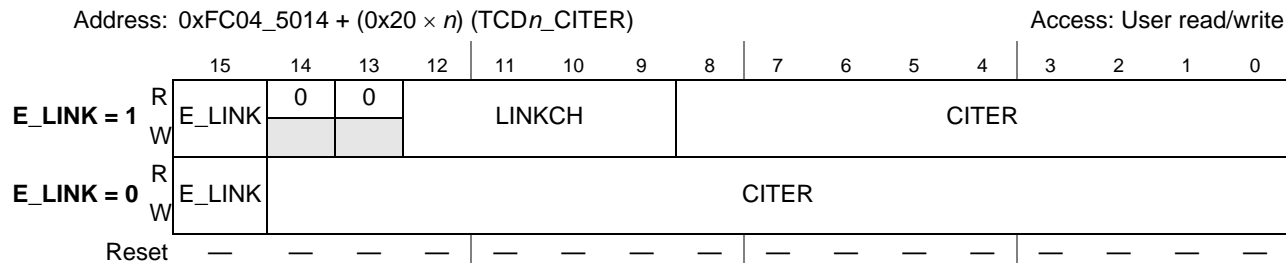
Figure 16-24. TCD*n* Current Major Iteration Count (TCD*n*_CITER)

Table 16-26. 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) (TCDn_DOFF) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								DOFF								
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
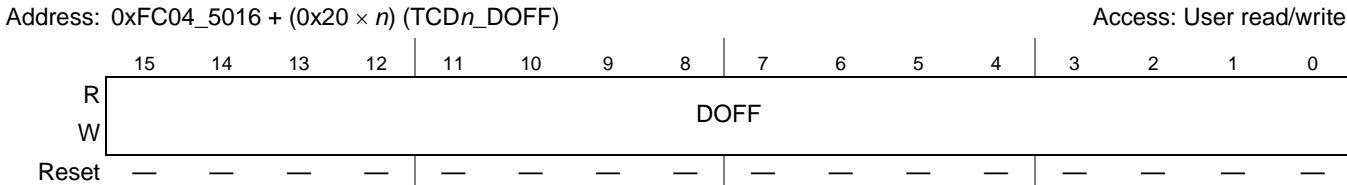
Figure 16-25. TCDn Destination Address Signed Offset (TCDn_DOFF)

Table 16-27. TCDn_DOFF Field Descriptions

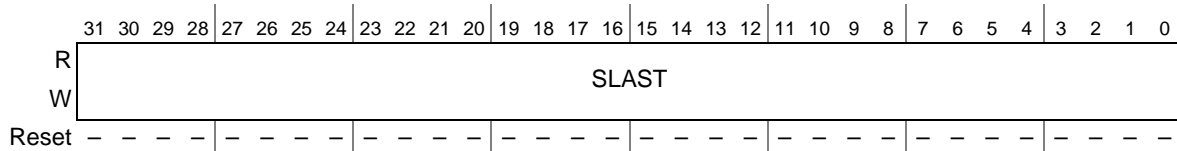| Field	Description
15–0 DOFF	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

Address: 0xFC04_5018 + (0x20 × n) (TCDn_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				DLAST_SGA				
W								
Reset	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –	– – – –
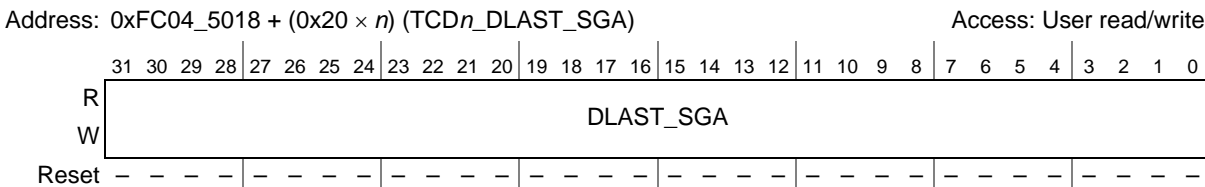
Figure 16-26. TCDn Destination Last Address Adjustment (TCDn_DLAST_SGA)

Table 16-28. TCDn_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 (TCDn_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) (TCDn_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/W	E_LINK	0	0		LINKCH			BITER								
E_LINK = 0	R/W	E_LINK							BITER								
	Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
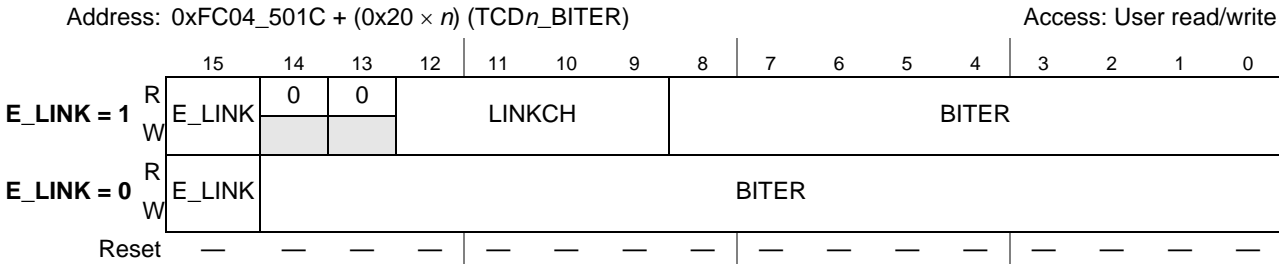
Figure 16-27. TCDn Beginning Major Iteration Count (TCDn_BITER)

Table 16-29. 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
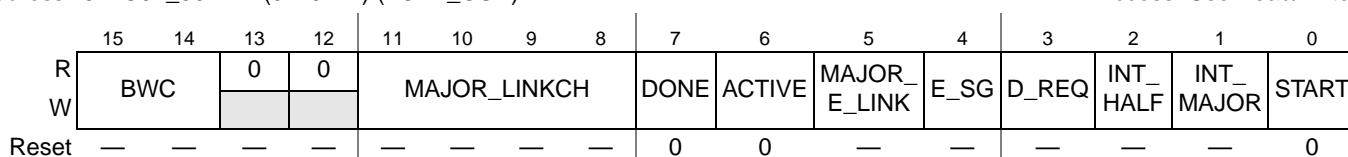
Figure 16-28. TCD*n* Control and Status (TCD*n*_CSR)

Table 16-30. 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 DMAERQ bit when the current major iteration count reaches zero. 0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the major loop is complete.

Table 16-30. 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.

16.7 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

16.7.1 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.

16.7.2 eDMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 16-29, 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.
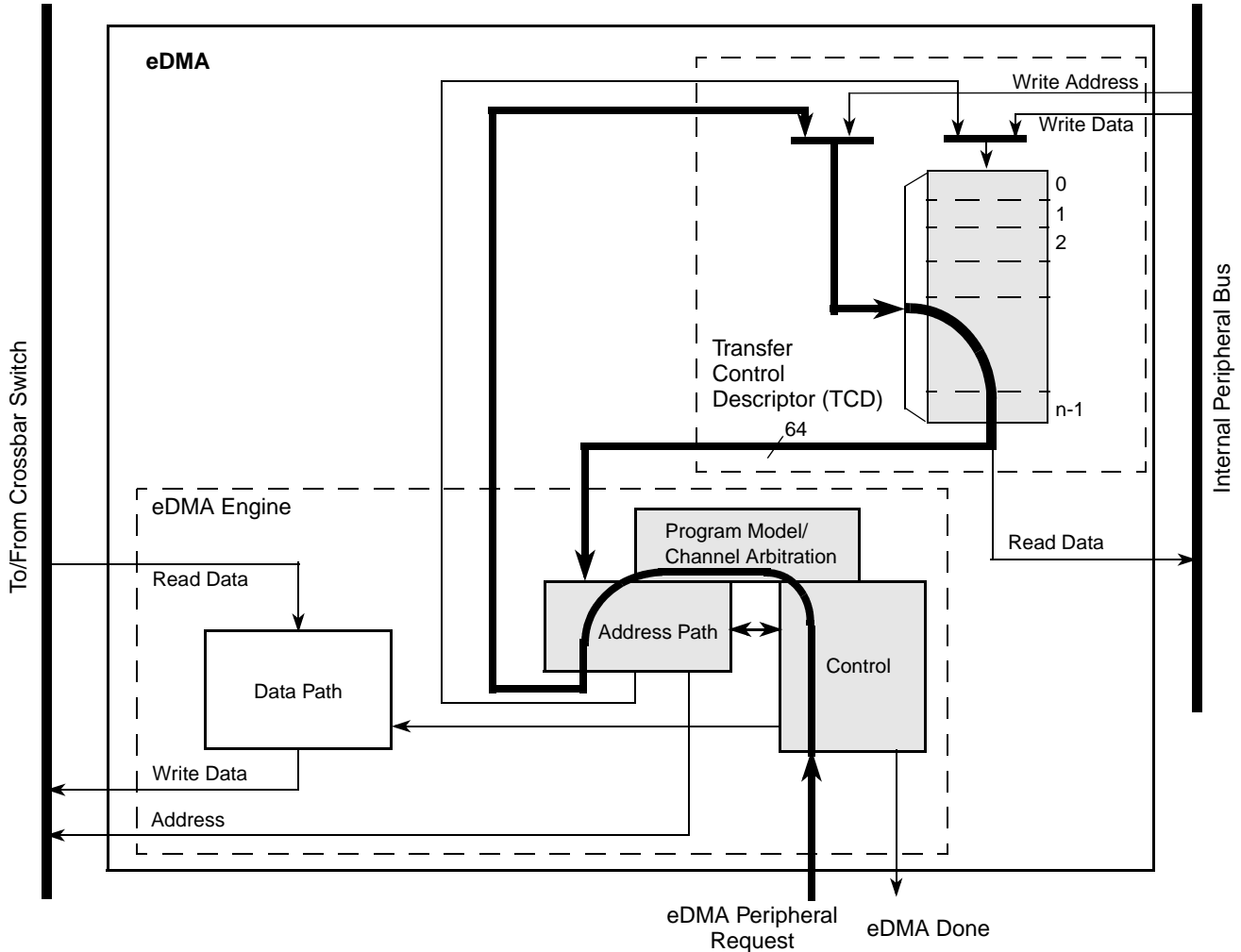
Figure 16-29. eDMA Operation, Part 1

In the second part of the basic data flow (Figure 16-30), 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.
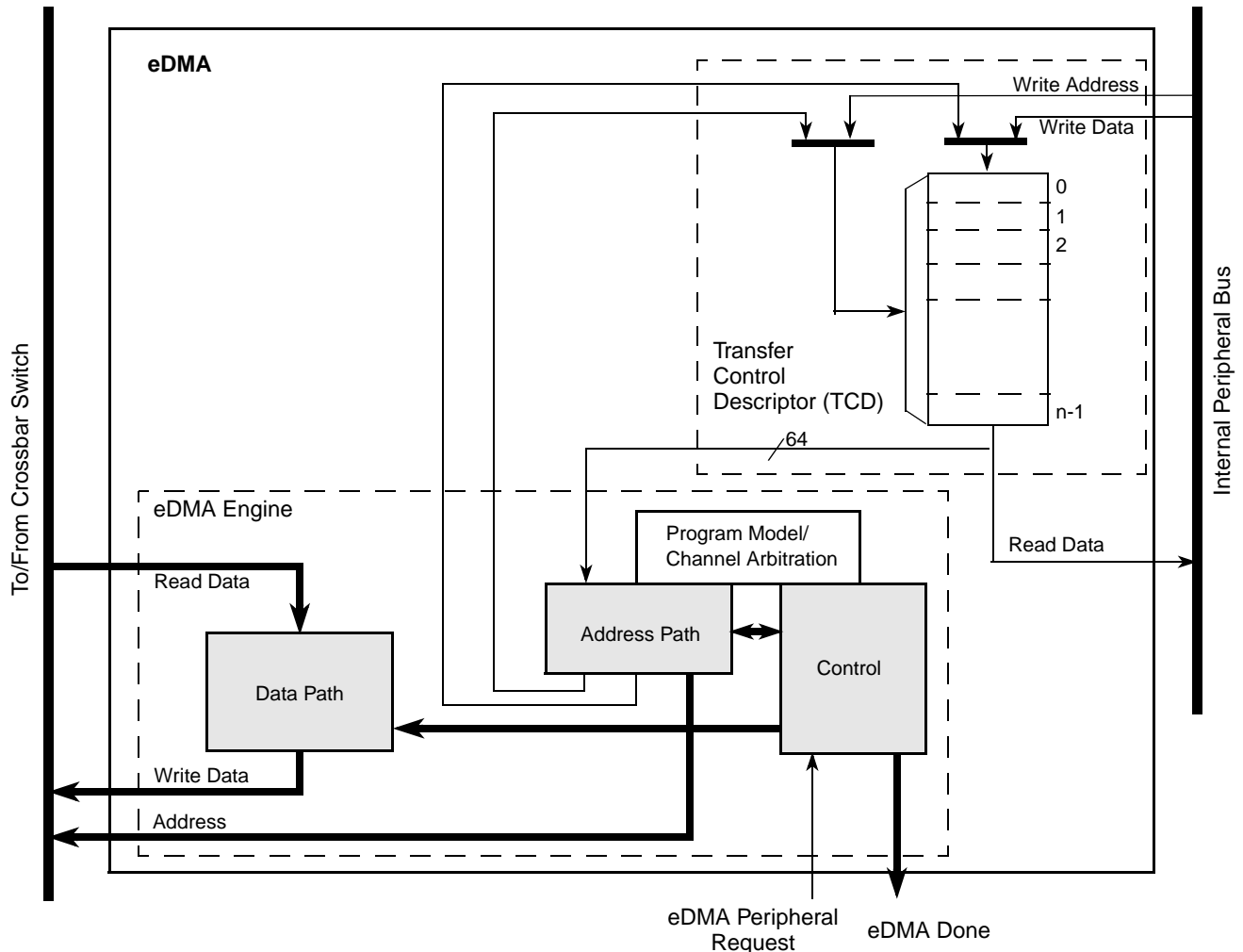
Figure 16-30. eDMA Operation, Part 2

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 16-31.
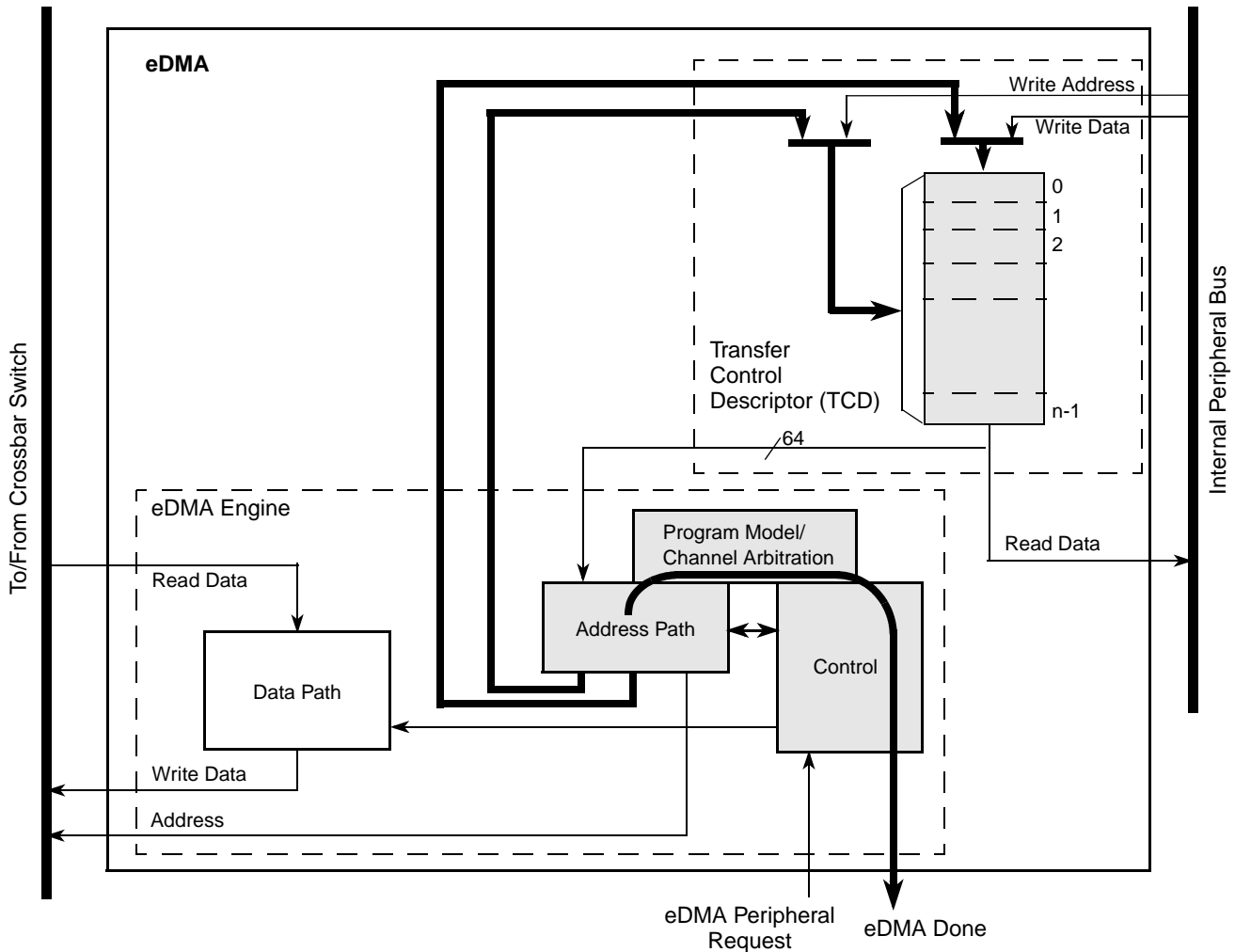
Figure 16-31. eDMA Operation, Part 3

16.8 Initialization/Application Information

16.8.1 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 16-31) 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 16-31. 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 16-32 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 16-32. Example of Multiple Loop Iterations

			Current Major Loop Iteration Count (CITER)
DMA Request	Minor Loop		3
DMA Request	Minor Loop	Major Loop	2
DMA Request	Minor Loop		1

Table 16-33 lists the memory array terms and how the TCD settings interrelate.

Table 16-33. 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	
*x*LAST: Number of bytes added to current address after major loop (typically used to loop back)	. . .	Last Minor Loop	

16.8.2 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.

16.8.3 DMA Arbitration Mode Considerations

16.8.3.1 Fixed Channel Arbitration

In this mode, the channel service request from the highest priority channel is selected to execute.

16.8.3.2 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

16.8.4 DMA Transfer

16.8.4.1 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 16-1. 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.

16.8.4.2 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.

16.8.4.3 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 16-34 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 16-34. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

16.8.5   eDMA TCD*n* Status Monitoring

16.8.5.1   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.

16.8.5.2   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.

16.8.5.3   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.

16.8.6   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 (TCDn_CITER[E_LINK] = 1), the TCDn_CITER[CITER] field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled (TCDn_CITER[E_LINK] = 0), the TCDn_CITER[CITER] field uses a 15-bit vector to form the current iteration count. The bits associated with the TCDn_CITER[LINKCH] field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

The TCDn_CITER[E_LINK] bit and the TCDn_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 16-35 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 16-35. 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

16.8.7   Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

16.8.7.1   Dynamic Channel Linking and Dynamic Scatter/Gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCDn_CSR[MAJOR_E_LINK] or TCDn_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 17
FlexBus

17.1 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 ($f_{sys/3}$).

The external data bus is shared between the FlexBus module and the SDRAM controller. When the SDRAM controller is in SDR mode (DRAMSEL = 1), the data bus is switched dynamically between the SDRAM controller and the FlexBus module. However, when the SDRAM controller is in DDR mode (DRAMSEL = 0), D[31:16] is dedicated to the SDRAM data bus and D[15:0] is dedicated to the FlexBus data bus. In this case, external pins D[15:0], are mapped internally to the upper two bytes of the FlexBus data bus, FB_D[31:16]. This chapter only uses FB_D[31:0] or FB_D[31:*X*] to designate the data bus, but the actual pins used depend on the DRAMSEL setting. Take this into consideration throughout this chapter.

17.1.1 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 with a maximum bus frequency up to 80 MHz. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry:

- External boot ROMs
- Flash memories
- Gate-array logic
- 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, $\overline{FB\_CS}$[5:0]. The actual number of chip selects available depends upon the device and its pin configuration. See Table 13-1 for more details. Chip-select $\overline{FB\_CS0}$ 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.

17.1.2 Features

Key FlexBus features include:

- Six independent, user-programmable chip-select signals ($\overline{FB\_CS}$[5:0]) that can interface with external SRAM, PROM, EPROM, EEPROM, flash, and other peripherals
- 8-, 16-, and 32-bit port sizes
- 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

17.2 External Signals

This section describes the external signals involved in data-transfer operations.

Table 17-1. FlexBus Signal Summary

Signal Name	I/O[1]	Description
FB_A[23:0]	O	Address bus. During the first cycle, this bus drives the upper address byte, addr[31:24].
FB_D[31:0]	I/O	Data bus
$\overline{FB\_CS}$[5:0]	O	General purpose chip-selects. The actual number of chip selects available depends upon the device and its pin configuration. See Table 13-1 for more details.
$\overline{FB\_BE/BWE}$[3:0]	O	Byte enable/byte write enable
$\overline{FB\_OE}$	O	Output enable
FB_R/$\overline{W}$	O	Read/write. 1 = Read, 0 = Write
$\overline{FB\_TS}$	O	Transfer start
$\overline{FB\_TA}$	I	Transfer acknowledge

[1] Because this device shares the FlexBus signals with the SDRAM controller, these signal directions are only valid when the FlexBus controls them. The directions may change during SDRAM cycles.

17.2.1 Address and Data Buses (FB_A[23:0], FB_D[31:0])

The FB_A[23:0] and FB_D[31:0] 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.

Because this device shares the FlexBus signals with the SDRAM controller, these signals tristate between bus cycles.

17.2.2 Chip Selects ($\overline{\text{FB\_CS}}$[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. See Table 13-1 for more details.

17.2.3 Byte Enables/Byte Write Enables ($\overline{\text{FB\_BE/BWE}}$[3:0])

When driven low, the byte enable ($\overline{\text{FB\_BE/BWE}}$[3:0]) outputs indicate data is to be latched or driven onto a byte of the data bus. $\overline{\text{FB\_BE/BWE}}n$ 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 $\overline{\text{FB\_BE/BWE}}n$ signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM or cache. For external SRAM or flash devices, the $\overline{\text{FB\_BE/BWE}}n$ outputs must be connected to individual byte strobe signals.

17.2.4 Output Enable ($\overline{\text{FB\_OE}}$)

The output enable signal ($\overline{\text{FB\_OE}}$) is sent to the interfacing memory and/or peripheral to enable a read transfer. $\overline{\text{FB\_OE}}$ is only asserted during read accesses when a chip select matches the current address decode.

Because this device shares the FlexBus signals with the SDRAM controller, this signal tristates between bus cycles.

17.2.5 Read/Write (FB_R/$\overline{\text{W}}$)

The processor drives the FB_R/$\overline{\text{W}}$ 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 SDRAM controller, this signal tristates between bus cycles.

17.2.6 Transfer Start ($\overline{\text{FB\_TS}}$)

The assertion of $\overline{\text{FB\_TS}}$ indicates that the device has begun a bus transaction and the address and attributes are valid. $\overline{\text{FB\_TS}}$ is asserted for one bus clock cycle.

Because this device shares the FlexBus signals with the SDRAM controller, this signal tristates between bus cycles.

17.2.7 Transfer Acknowledge ($\overline{\text{FB\_TA}}$)

This signal indicates the external data transfer is complete. When the processor recognizes $\overline{\text{FB\_TA}}$ during a read cycle, it latches the data and then terminates the bus cycle. When the processor recognizes $\overline{\text{FB\_TA}}$ during a write cycle, the bus cycle is terminated.

If auto-acknowledge is disabled (CSCRn[AA] = 0), the external device drives $\overline{FB\_TA}$ to terminate the bus transfer; if auto-acknowledge is enabled (CSCRn[AA] = 1), $\overline{FB\_TA}$ is generated internally after a specified number of wait states, or the external device may assert external $\overline{FB\_TA}$ before the wait-state countdown, terminating the cycle early. The device negates $\overline{FB\_CS}n$ one cycle after the last $\overline{FB\_TA}$ asserts. During read cycles, the peripheral must continue to drive data until $\overline{FB\_TA}$ is recognized. For write cycles, the processor continues driving data one clock after $\overline{FB\_CS}n$ is negated.

The number of wait states is determined by CSCRn or the external $\overline{FB\_TA}$ input. If the external $\overline{FB\_TA}$ is used, the peripheral has total control on the number of wait states.

NOTE

External devices should only assert $\overline{FB\_TA}$ while the $\overline{FB\_CS}n$ signal to the external device is asserted.

Because this device shares the FlexBus signals with the SDRAM controller, this signal tristates between bus cycles.

17.3 Memory Map/Register Definition

The following tables describe the registers and bit meanings for configuring chip-select operation. Table 17-2 shows the chip-select register memory map.

The actual number of chip select registers available depends upon the device and its pin configuration. See Table 13-1 for more details. 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.

Table 17-2. FlexBus Chip Select Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC00_8000 + ($n \times$ 0xC)	Chip-Select Address Register (CSARn) $n = 0 - 5$	32	R/W	0x0000_0000	17.3.1/17-4
0xFC00_8004 + ($n \times$ 0xC)	Chip-Select Mask Register (CSMRn) $n = 0 - 5$	32	R/W	0x0000_0000	17.3.2/17-5
0xFC00_8008 + ($n \times$ 0xC)	Chip-Select Control Register (CSCRn) $n = 0 - 5$	32	R/W	See Section	17.3.3/17-6

17.3.1 Chip-Select Address Registers (CSAR0 – CSAR5)

The CSARn 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 CSARn registers appropriately.

Address: 0xFC00_8000 (CSAR0)   Access: User read/write
    0xFC00_800C (CSAR1)
    0xFC00_8018 (CSAR2)
    0xFC00_8024 (CSAR3)
    0xFC00_8030 (CSAR4)
    0xFC00_803C (CSAR5)

	31	30	29	28	27	26	25	24	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
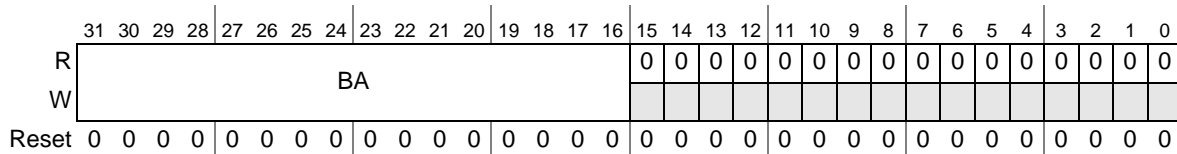
Figure 17-1. Chip-Select Address Registers (CSAR*n*)

Table 17-3. CSAR*n* Field Descriptions

Field	Description
31–16 BA	Base address. Defines the base address for memory dedicated to chip-select $\overline{FB\_CS}$*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.

17.3.2 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)   Access: User read/write
    0xFC00_8010 (CSMR1)
    0xFC00_801C (CSMR2)
    0xFC00_8028 (CSMR3)
    0xFC00_8034 (CSMR4)
    0xFC00_8040 (CSMR5)

	31	30	29	28	27	26	25	24	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
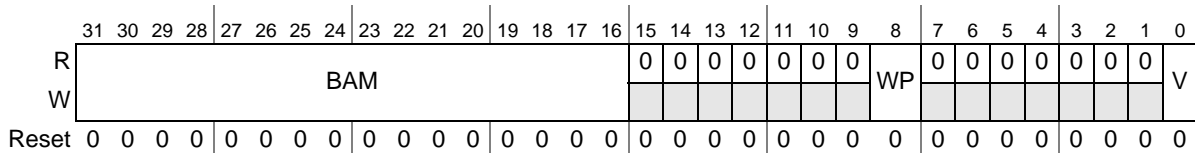
Figure 17-2. Chip-Select Mask Registers (CSMR*n*)

MCF5329 Reference Manual, Rev 3

Freescale Semiconductor   17-5

Table 17-4. 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 $\overline{FB\_CSn}$ is 2^n; n = (number of bits set in respective CSMR[BAM]) + 16. For example, if CSAR0 equals 0x0000 and CSMR0[BAM] equals 0x0008, $\overline{FB\_CS0}$ addresses two discontinuous 64-Kbyte memory blocks: one from 0x0_0000 – 0x0_FFFF and one from 0x8_0000 – 0x8_FFFF. Likewise, for $\overline{FB\_CS0}$ to access 32 Mbytes of address space starting at location 0x00_0000, $\overline{FB\_CS1}$ must begin at the next byte after $\overline{FB\_CS0}$ for a 16-Mbyte address space. Then, 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 $\overline{FB\_CS0}$, which acts as the global chip-select). Reset clears each CSMR*n*[V]. **Note:** At reset, no chip-select other than $\overline{FB\_CS0}$ can be used until the CSMR0[V] is set. Afterward, $\overline{FB\_CS}$[5:0] functions as programmed. 0 Chip-select invalid 1 Chip-select valid

17.3.3   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, $\overline{FB\_CS0}$, the CSCR0 reset values differ from the

other CSCRs. $\overline{FB\_CS0}$ 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				SBM	AA	PS		BEM	BSTR	BSTW	0	0	0
W																
Reset: CSCR0	1	1	1	1	1	1	[$\overline{DRAM\ SEL}$]	1	D4	D3	1	0	0	0	0	0
Reset: CSCR1–5	0	0	0	0	0	0	[$\overline{DRAM\ SEL}$]	1	0	0	1	0	0	0	0	0

Figure 17-3. Chip-Select Control Registers (CSCR*n*)

Table 17-5. 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 $\overline{FB\_TS}$ asserts. 00　Assert $\overline{FB\_CSn}$ on first rising clock edge after address is asserted. (Default $\overline{FB\_CSn}$) 01　Assert $\overline{FB\_CSn}$ on second rising clock edge after address is asserted. 10　Assert $\overline{FB\_CSn}$ on third rising clock edge after address is asserted. 11　Assert $\overline{FB\_CSn}$ on fourth rising clock edge after address is asserted. (Default $\overline{FB\_CS0}$)

Table 17-5. CSCR*n* Field Descriptions (continued)

Field	Description
19–18 RDAH	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. 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 $\overline{FB\_CSn}$ negation depends on the value of CSCR*n*[AA] as shown below. <table><tr><th>RDAH</th><th>AA = 0</th><th>AA = 1</th></tr><tr><td>00 ($\overline{FB\_CSn}$ 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 ($\overline{FB\_CS0}$ Default)</td><td>4 cycles</td><td>3 cycles</td></tr></table>
17–16 WRAH	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. 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 $\overline{FB\_CSn}$ negates on writes. (Default $\overline{FB\_CSn}$) 01 Hold address and attributes two cycles after $\overline{FB\_CSn}$ negates on writes. 10 Hold address and attributes three cycles after $\overline{FB\_CSn}$ negates on writes. 11 Hold address and attributes four cycles after $\overline{FB\_CSn}$ negates on writes. (Default $\overline{FB\_CS0}$)
15–10 WS	Wait states. The number of wait states inserted after $\overline{FB\_CSn}$ 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, $\overline{FB\_TA}$ 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 $\overline{FB\_TA}$ supersedes the generation of an internal $\overline{FB\_TA}$.
9 SBM	Split bus mode. For proper operation of the chip select signals, this bit must be set when the SDRAM controller is in DDR mode (DRAMSEL signal is negated). The reset value of SBM is the opposite of the DRAMSEL signal. 0 Device is not in split bus mode (SDRAM controller is in SDR mode, DRAMSEL = 1). 1 Device is in split bus mode (SDRAM controller is in DDR mode, DRAMSEL = 0). **Note:** Placing the device in split bus mode is only controlled by the DRAMSEL signal. This bit is only used to provide correct operation of the chip select signals.
8 AA	Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address. 0 No internal $\overline{FB\_TA}$ is asserted. Cycle is terminated externally 1 Internal transfer acknowledge is asserted as specified by WS **Note:** If AA is set for a corresponding $\overline{FB\_CSn}$ and the external system asserts an external $\overline{FB\_TA}$ before the wait-state countdown asserts the internal $\overline{FB\_TA}$, the cycle is terminated. Burst cycles increment the address bus between each internal termination. **Note:**
7–6 PS	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] if SBM = 0 or FB_D[7:0] if SBM = 1 1*x* 16-bit port size. Valid data sampled and driven on FB_D[31:16] if SBM = 0 or FB_D[15:0] if SBM = 1

Table 17-5. 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 $\overline{\text{FB\_BE/BWE}}$ is not asserted for reads. $\overline{\text{FB\_BE/BWE}}$ is asserted for data write only. 1 $\overline{\text{FB\_BE/BWE}}$ is asserted for read and write accesses.
4 BSTR	Burst-read enable. Specifies whether burst reads are used for memory associated with each $\overline{\text{FB\_CS}n}$. 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 $\overline{\text{FB\_CS}n}$. 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.

17.4 Functional Description

17.4.1 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 17.3.1, "Chip-Select Address Registers (CSAR0 – CSAR5)."

- Chip-select mask registers (CSMR*n*) provide 16-bit address masking and access control. See Section 17.3.2, "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 17.3.3, "Chip-Select Control Registers (CSCR0 – CSCR5)."

$\overline{\text{FB\_CS0}}$ is a global chip-select after reset and provides external boot memory capability.

17.4.1.1 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 17-6.

Table 17-6. 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.
No	The internal bus cycle terminates and no chip select is asserted.
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.

17.4.1.2 8-, 16-, and 32-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. The processor always drives a 24-bit address on the FB_A bus regardless of the external device's address size. The external device must connect its address lines to the appropriate FB_A bits from FB_A0 upward. It must also connect its data lines to the FB_D bus from FB_D31 downward. No bit ordering is required when connecting address and data lines to the FB_A and FB_D buses. For example, a full 16-bit address/16-bit data device connects its addr[15:0] to FB_A[16:1] and data[15:0] to FB_D[31:16]. See Figure 17-4 for a graphical connection.

17.4.1.3 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, the logic levels on the FB_D[4:3] signals determine global chip-select port size.

 See Chapter 9, "Chip Configuration Module (CCM)," for more information.

17.4.2 Data Transfer Operation

Data transfers between the chip and other devices involve these signals:

- Address/data bus (FB_A[23:0], FB_D[31:0])
- Control signals ($\overline{\text{FB\_TS}}$, $\overline{\text{FB\_TA}}$, $\overline{\text{FB\_CS}n}$, $\overline{\text{FB\_OE}}$, $\overline{\text{FB\_BE/BWE}}$[3:0])
- Attribute signals (FB_R/$\overline{\text{W}}$)

The address, write data, $\overline{\text{FB\_TS}}$, $\overline{\text{FB\_CS}n}$, 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 17.3.3, "Chip-Select Control Registers (CSCR0 – CSCR5)."

17.4.3   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. The byte lane assignment is also dependent on the split bus mode setting in the CSCR*n* register.

Figure 17-4 shows the byte lanes that external memory connects to and the sequential transfers of a longword transfer for the supported port sizes when not in split bus mode. For example, an 8-bit memory connects to the single lane FB_D[31:24] ($\overline{\text{FB\_BE/BWE0}}$). 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.
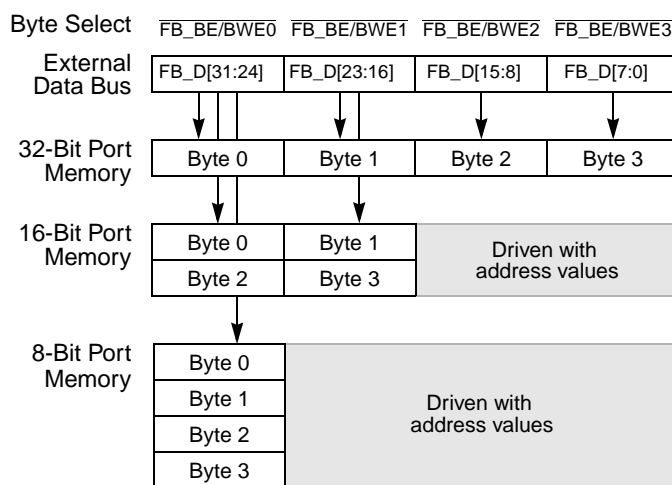
Figure 17-4. Connections for External Memory Port Sizes (CSCR*n*[SBM] = 0)
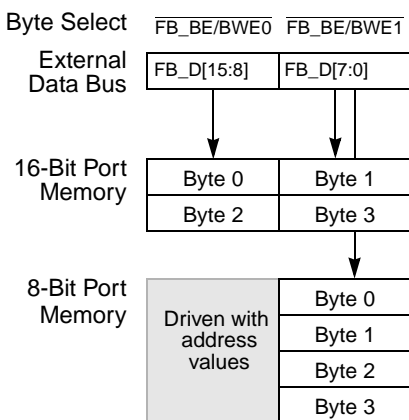
Figure 17-5. Connections for External Memory Port Sizes (CSCR*n*[SBM] = 1)

17.4.4 Bus Cycle Execution

As shown in Figure 17-8 and Figure 17-10, basic bus operations occur in four clocks:

1. S0: At the first clock edge, the address, attributes, and $\overline{FB\_TS}$ are driven.

2. S1: $\overline{FB\_CSn}$ is asserted at the second rising clock edge to indicate the device selected; by that time, the address and attributes are valid and stable. $\overline{FB\_TS}$ 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 $\overline{FB\_CSn}$ negates. For a read transfer, data is also driven into the device during this cycle.

   External slave asserts $\overline{FB\_TA}$ at this clock edge.

3. S2: Read data and $\overline{FB\_TA}$ are sampled on the third clock edge. $\overline{FB\_TA}$ can be negated after this edge and read data can then be tri-stated.

4. S3: $\overline{FB\_CSn}$ 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.

17.4.4.1 Data Transfer Cycle States

An on-chip state machine controls the data-transfer operation in the device. Figure 17-6 shows the state-transition diagram for basic read and write cycles.
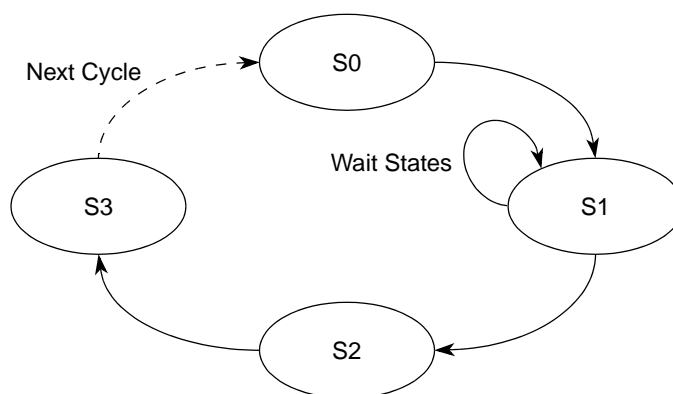
Figure 17-6. Data-Transfer-State-Transition Diagram

Table 17-7 describes the states as they appear in subsequent timing diagrams.

Table 17-7. 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_A[23:0], asserts $\overline{FB\_TS}$, and drives FB_R/$\overline{W}$ high for a read and low for a write.

Table 17-7. Bus Cycle States (continued)

State	Cycle	Description
S1	All	$\overline{FB\_TS}$ is negated on the rising edge of FB_CLK, and $\overline{FB\_CSn}$ is asserted. Data is driven on FB_D[31:*X*] for writes, and FB_D[31:*X*] is tristated for reads. Address continues to be driven on the FB_A pins. If $\overline{FB\_TA}$ is recognized asserted, then the cycle moves on to S2. If $\overline{FB\_TA}$ 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 $\overline{FB\_TA}$ asserted.
S2	All	For internal termination, $\overline{FB\_CSn}$ is negated and the internal system bus transfer is completed. For external termination, the external device should negate $\overline{FB\_TA}$, and the $\overline{FB\_CSn}$ 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/$\overline{W}$ go invalid off the rising edge of FB_CLK at the beginning of S3, terminating the read or write cycle.

17.4.5 FlexBus Timing Examples

NOTE

Because this device shares the FlexBus signals with the SDRAM controller, all signals, except the chip selects, tristate between bus cycles.

17.4.5.1 Basic Read Bus Cycle

During a read cycle, the ColdFire device receives data from memory or a peripheral device. Figure 17-7 is a read cycle flowchart.

NOTE

Throughout this chapter FB_D[31:*X*] indicates a 32-, 16-, or 8-bit wide data bus.
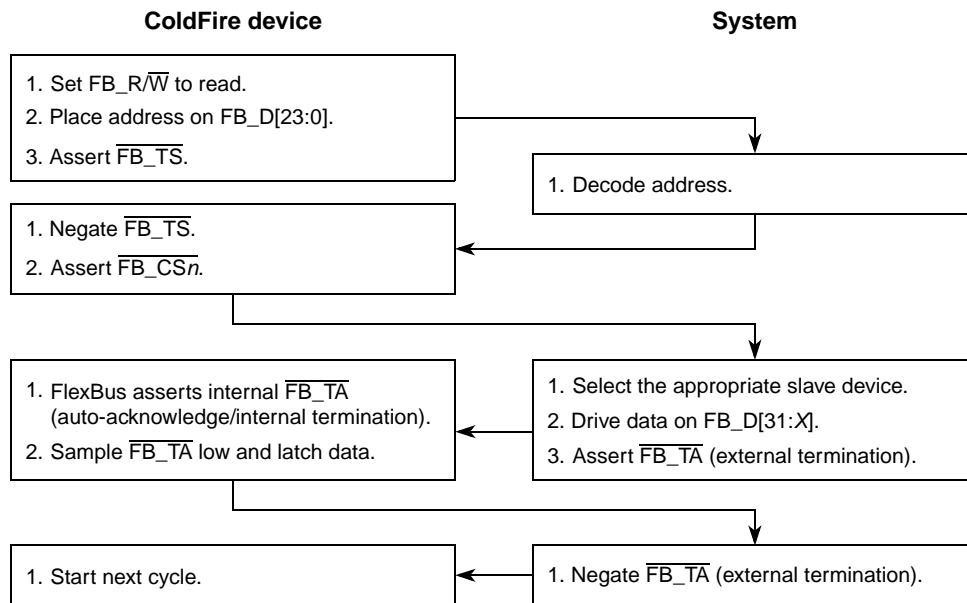
Figure 17-7. Read Cycle Flowchart

The read cycle timing diagram is shown in Figure 17-8.

NOTE

In the next set of timing diagrams, the dotted lines indicate $\overline{FB\_TA}$, $\overline{FB\_OE}$, and $\overline{FB\_CSn}$ timing when internal termination is used (CSCR[AA] = 1). The external and internal $\overline{FB\_TA}$ assert at the same time; however, $\overline{FB\_TA}$ 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 feature beneficial.

The address and data busses are muxed between the FlexBus and SDRAM controller. At the end of the read bus cycles the address signals are indeterminate.

S0 S1 S2 S3 S0

FB_CLK

FB_A[23:0]   ADDR[23:0]

FB_D[31:*X*]   ADDR[31:*X*]   DATA

FB_R/$\overline{W}$

$\overline{FB\_TS}$

$\overline{FB\_CSn}$, $\overline{FB\_OE}$,
$\overline{FB\_BE}$/BWE*n*

$\overline{FB\_TA}$
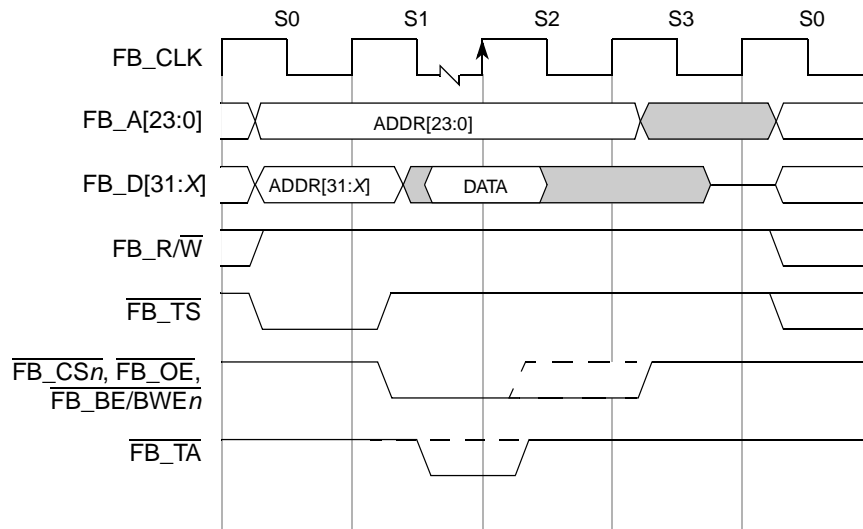
Figure 17-8. Basic Read-Bus Cycle

17.4.5.2 Basic Write Bus Cycle

During a write cycle, the device sends data to memory or to a peripheral device. Figure 17-9 shows the write cycle flowchart.

ColdFire device   **System**

1. Set FB_R/$\overline{W}$ to write.
2. Place address on FB_D[23:0].
3. Assert $\overline{FB\_TS}$.

1. Decode address.

1. Negate $\overline{FB\_TS}$.
2. Assert $\overline{FB\_CSn}$.
3. Drive data.

1. FlexBus asserts internal $\overline{FB\_TA}$ (auto acknowledge/internal termination).
2. Sample $\overline{FB\_TA}$ low.

1. Select the appropriate slave device.
2. Latch data on FB_D[31:*X*].
3. Assert $\overline{FB\_TA}$ (external termination).

1. Start next cycle.

1. Negate $\overline{FB\_TA}$ (external termination).

Figure 17-9. Write-Cycle Flowchart

Figure 17-10 shows the write cycle timing diagram.

NOTE

The address and data busses are muxed between the FlexBus and SDRAM controller. At the end of the write bus cycles, the address signals are indeterminate.
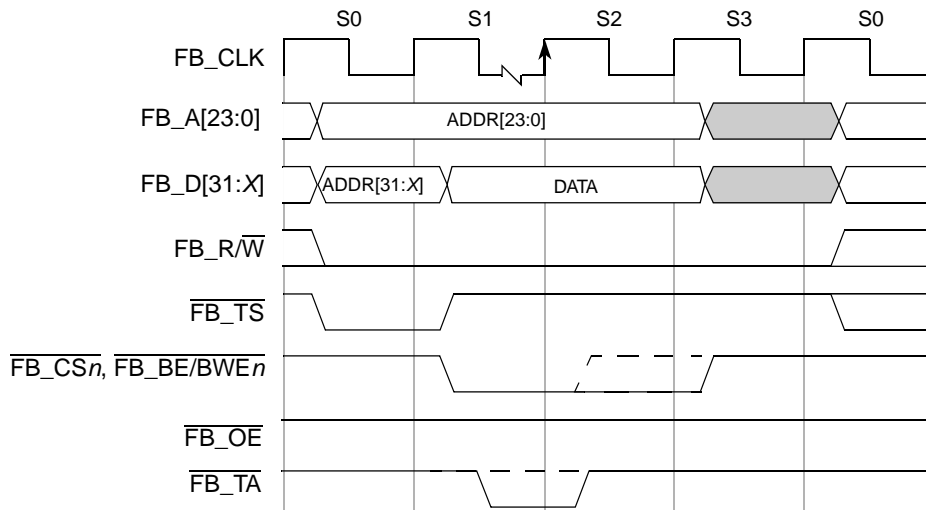
Figure 17-10. Basic Write-Bus Cycle

17.4.5.3 Bus Cycle Sizing

This section shows timing diagrams for various port size scenarios. Figure 17-11 illustrates the basic byte read transfer to an 8-bit device with no wait states. The address is driven on the FB_A[23:8] bus throughout the bus cycle. The external device returns the read data on FB_D[31:24] and may tristate the data line or continue driving the data one clock after FB_TA is sampled asserted.
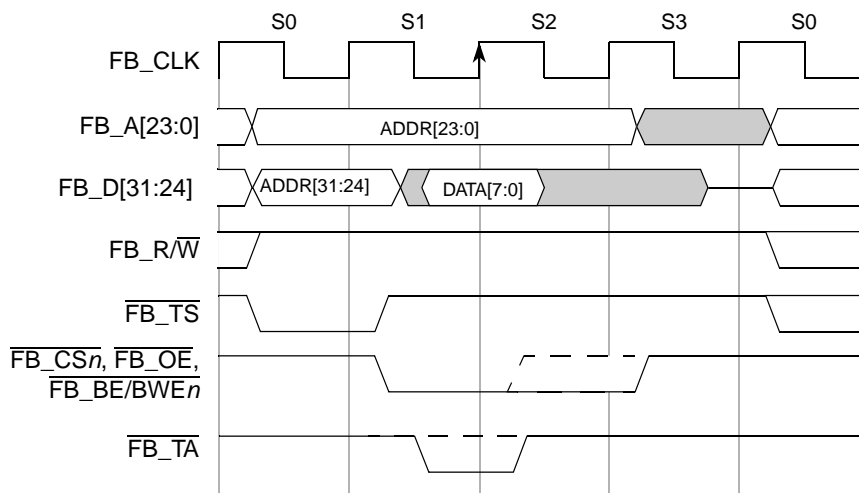
Figure 17-11. Single Byte-Read Transfer

Figure 17-12 shows the similar configuration for a write transfer. The data is driven from the second clock on FB_D[31:24].
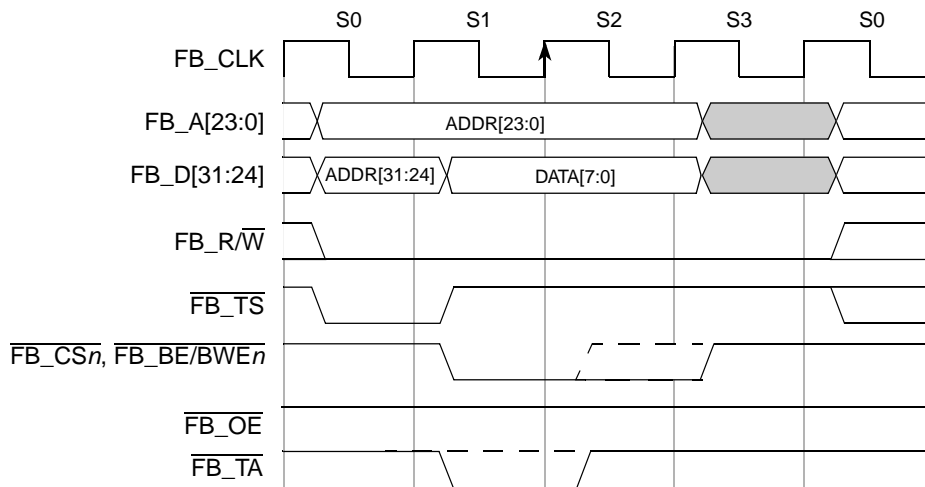
Figure 17-12. Single Byte-Write Transfer

Figure 17-13 illustrates the basic word read transfer to a 16-bit device with no wait states. The address is driven on the FB_A[23:8 :0] bus throughout the bus cycle. The external device returns the read data on FB_D[31:16], and may tristate the data line or continue driving the data one clock after FB_TA is sampled asserted.
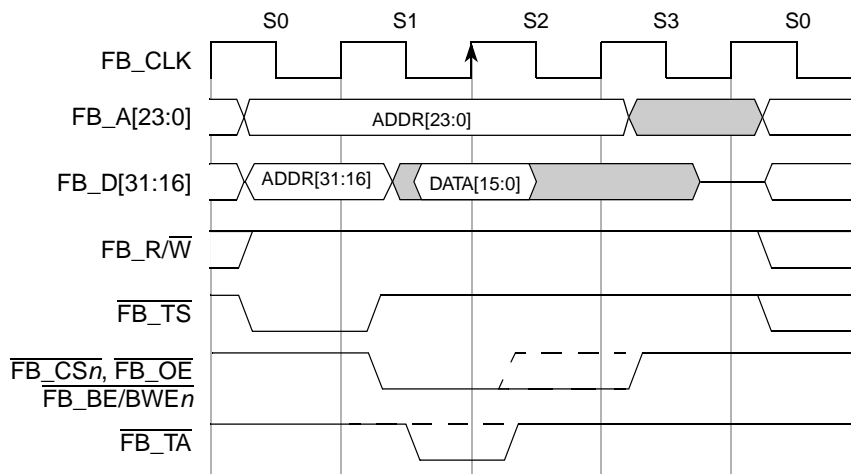
Figure 17-13. Single Word-Read Transfer

Figure 17-14 shows the similar configuration for a write transfer. The data is driven from the second clock on FB_D[31:16].
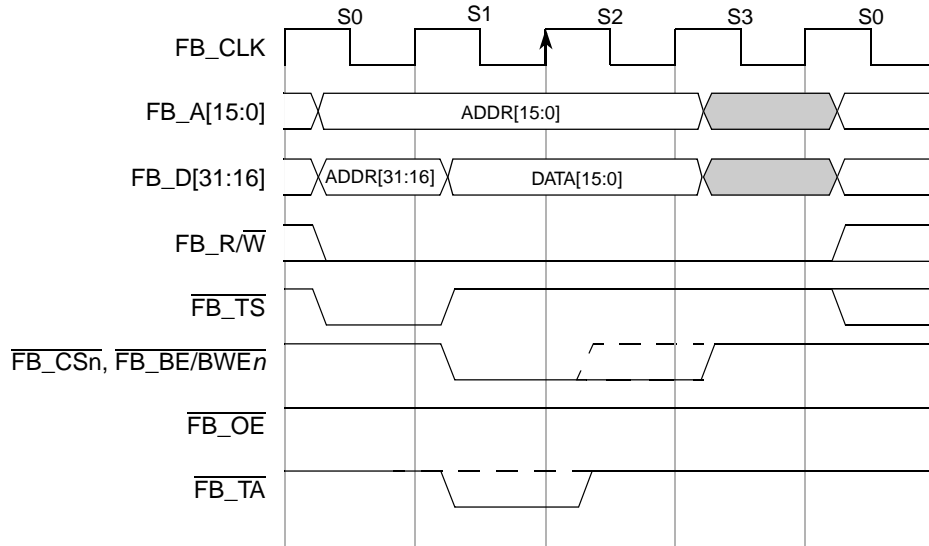
Figure 17-14. Single Word-Write Transfer

Figure 17-15 depicts a longword read from a 32-bit device.
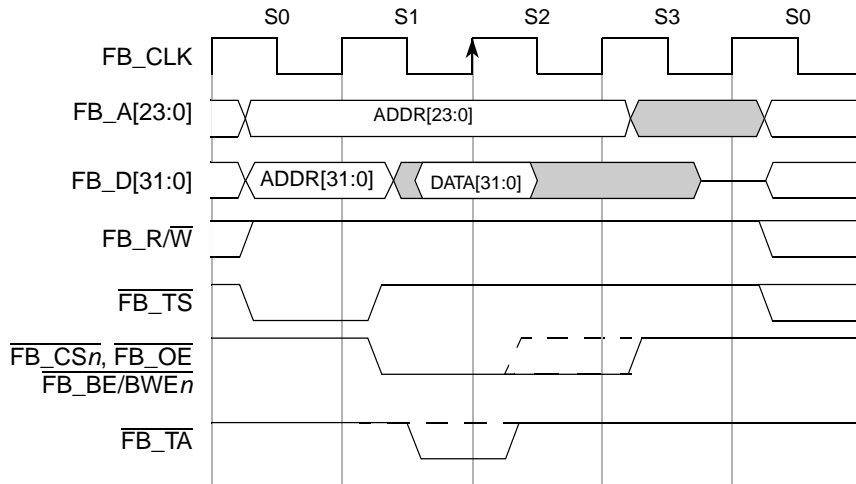
Figure 17-15. Longword-Read Transfer

Figure 17-16 illustrates the longword write to a 32-bit device.
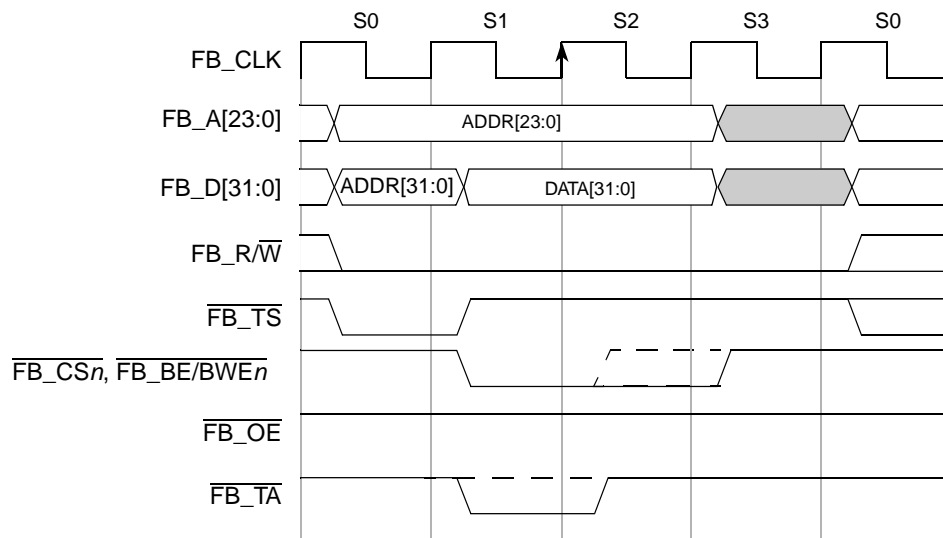
Figure 17-16. Longword-Write Transfer

17.4.5.4 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.

17.4.5.4.1 Wait States

Wait states can be inserted before each beat of a transfer by programming the CSCRn registers. Wait states can give the peripheral or memory more time to return read data or sample write data.

Figure 17-17 and Figure 17-18 show the basic read and write bus cycles (also shown in Figure 17-8 and Figure 17-13) with the default of no wait states.
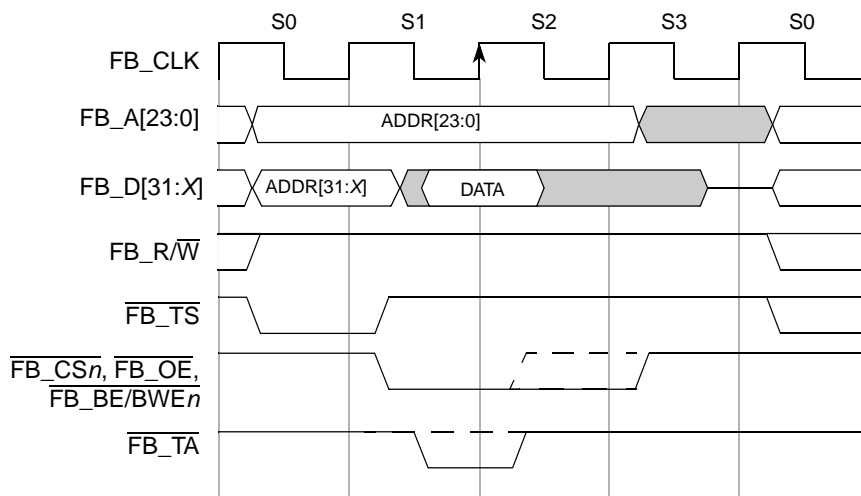
Figure 17-17. Basic Read-Bus Cycle (No Wait States)
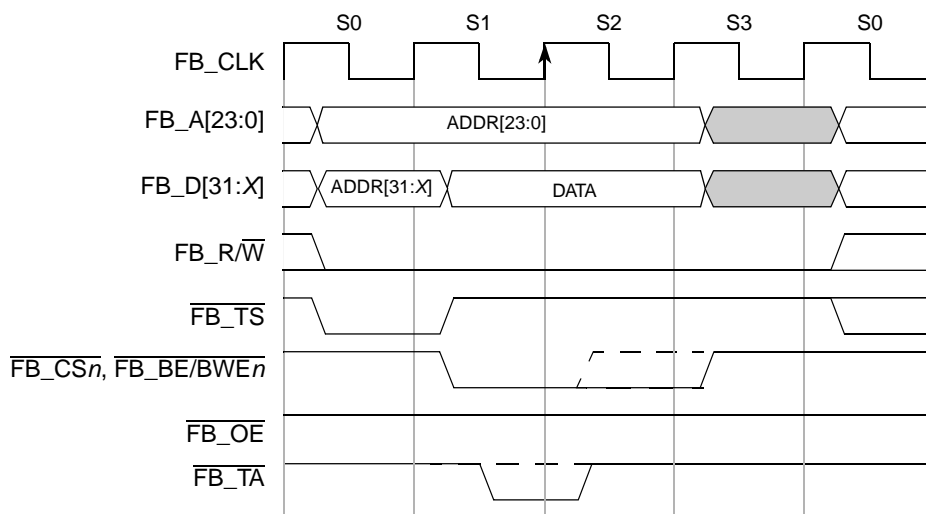
Figure 17-18. Basic Write-Bus Cycle (No Wait States)

If wait states are used, the S1 state repeats continuously until the the chip-select auto-acknowledge unit asserts internal transfer acknowledge or the external $\overline{FB\_TA}$ is recognized as asserted. Figure 17-19 and Figure 17-20 show a read and write cycle with one wait state.
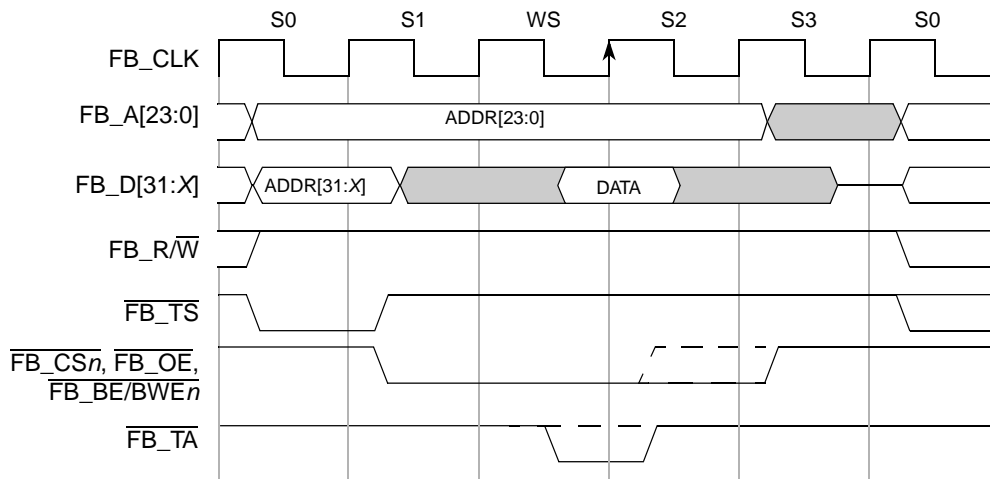
Figure 17-19. Read-Bus Cycle (One Wait State)
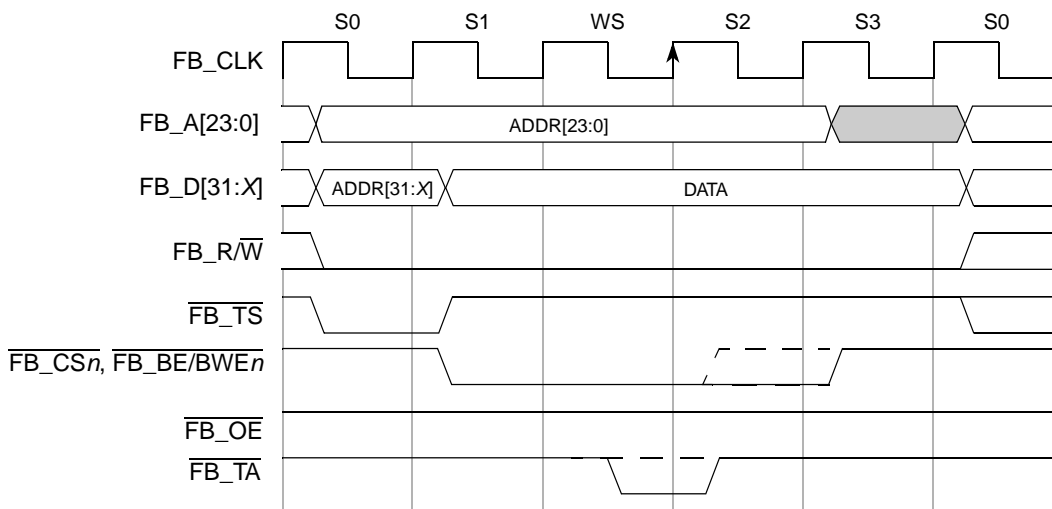
Figure 17-20. Write-Bus Cycle (One Wait State)

17.4.5.4.2   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

transfer start ($\overline{\text{FB\_TS}}$) is asserted. Figure 17-21 and Figure 17-22 show read- and write-bus cycles with two clocks of address setup.
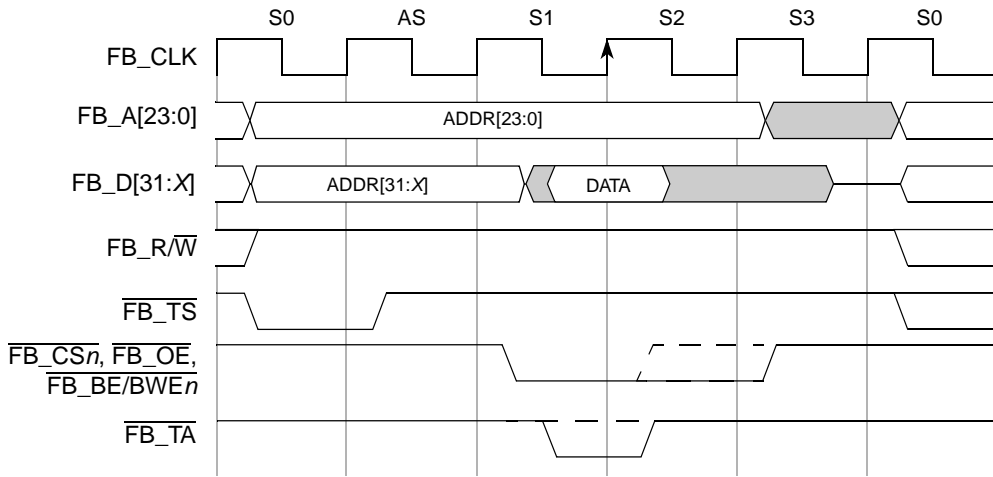
Figure 17-21. Read-Bus Cycle with Two-Clock Address Setup (No Wait States)
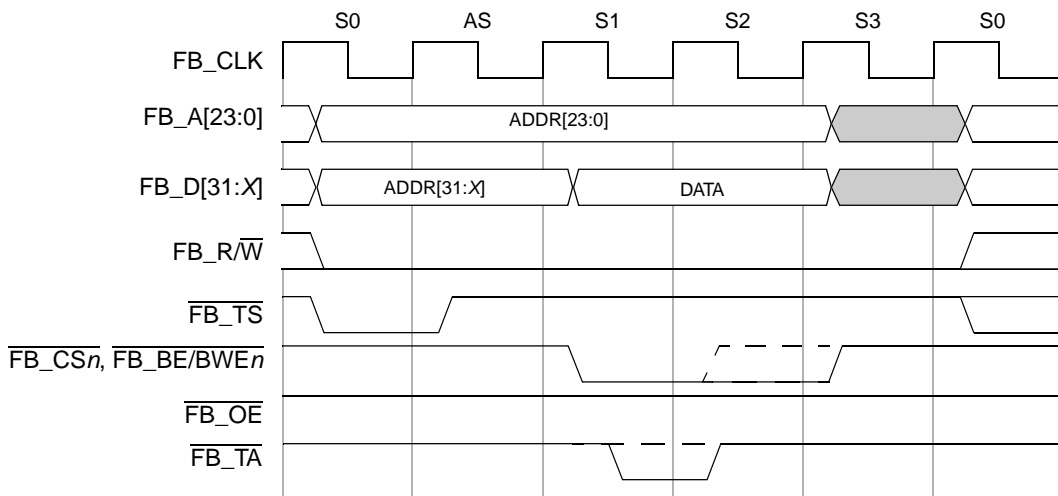
Figure 17-22. Write-Bus Cycle with Two Clock Address Setup (No Wait States)

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 17-23 and Figure 17-24 show read and write bus cycles with two clocks of address hold.
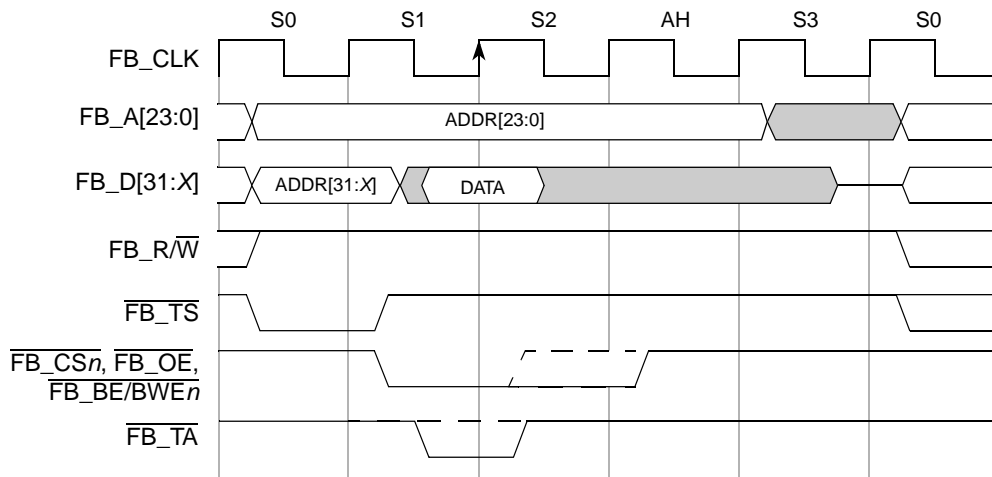
Figure 17-23. Read Cycle with Two-Clock Address Hold (No Wait States)
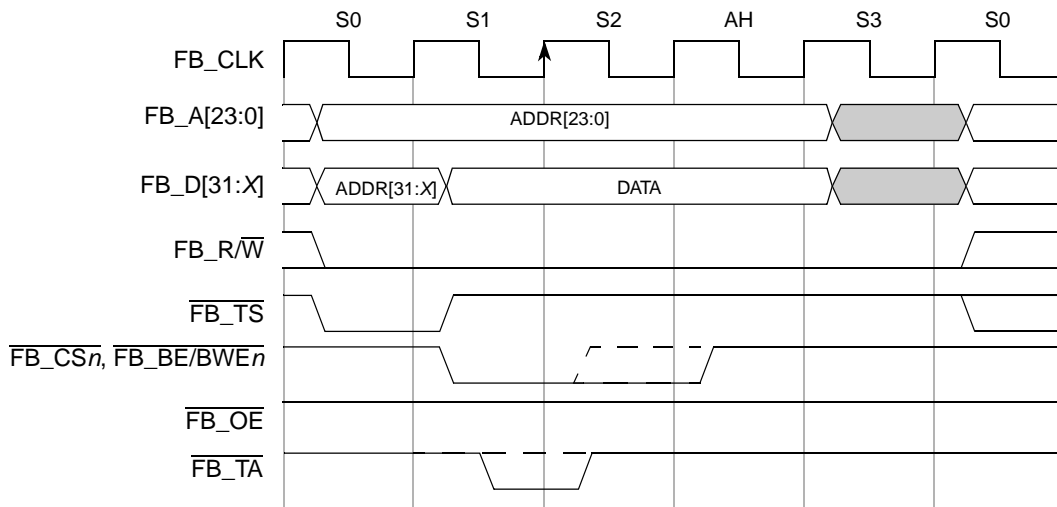
Figure 17-24. Write Cycle with Two-Clock Address Hold (No Wait States)

Figure 17-25 shows a bus cycle using address setup, wait states, and address hold.
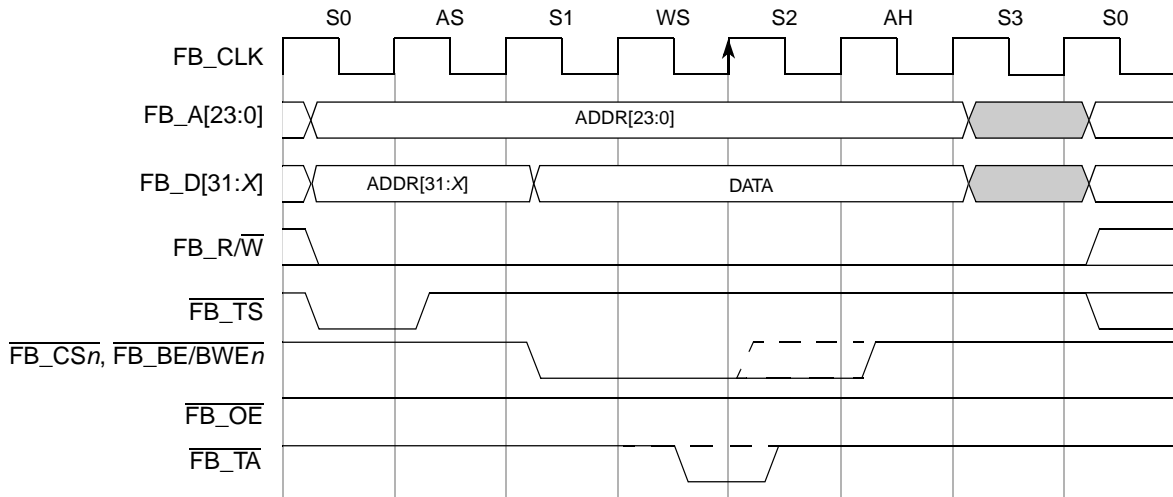
**Figure 17-25. Write Cycle with Two-Clock Address Setup and
Two-Clock Hold (One Wait State)**

17.4.6 Burst Cycles

The device can be programmed to initiate burst cycles if its transfer size exceeds the port size of the selected destination. 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 17-8 shows the result of such transfer translations.

Table 17-8. Transfer Size and Port Size Translation

Port Size PS[1:0]	Transfer Size	Burst-Inhibited: Number of Transfers Burst Enabled: Number of Beats
01 (8-bit)	word	2
	longword	4
	line	16
1x (16-bit)	longword	2
	line	8
00 (32-bit)	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 CSCRn registers enable bursting for reads, writes, or both. Memory spaces can be declared burst-inhibited for reads and writes by clearing the appropriate CSCRn[BSTR,BSTW] bits.

Figure 17-26 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_D[31:24].

NOTE

Address lines increment only during internally-terminated burst cycles. The first address is driven throughout the entire burst for externally-terminated cycles.
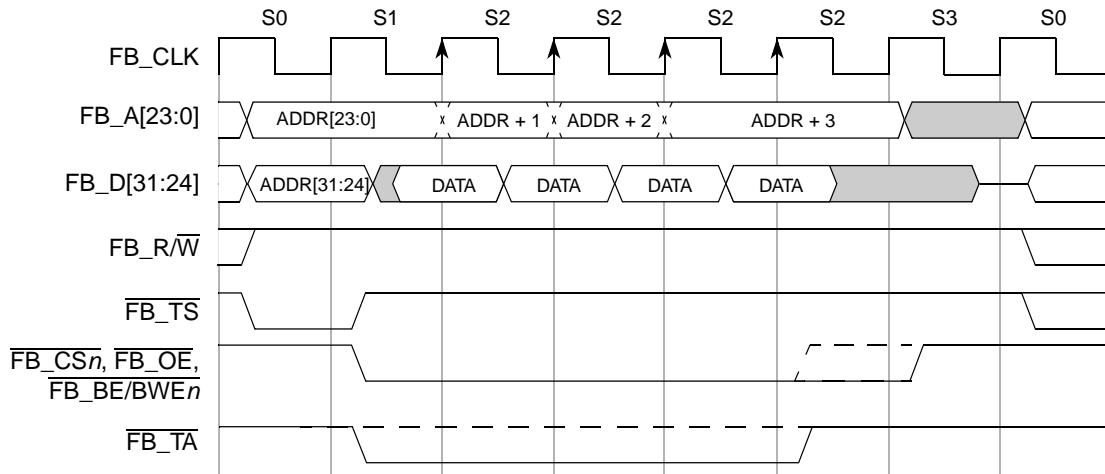
Figure 17-26. Longword-Read Burst from 8-Bit Port 2-1-1-1 (No Wait States)

Figure 17-27 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_D[31:24].

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 (CSCRn[WS] = 0), one wait state is added. Otherwise, the programmed number of wait states are used.
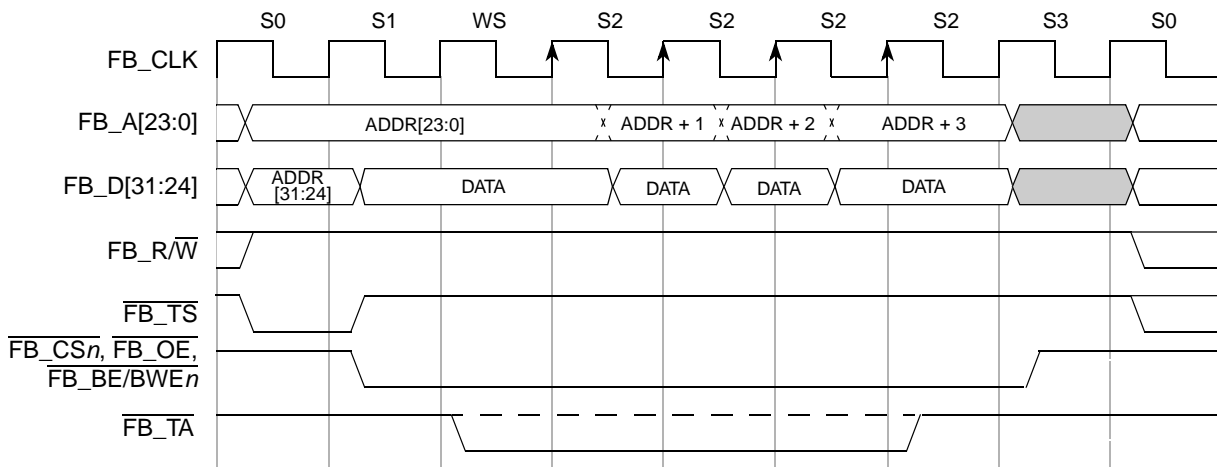
Figure 17-27. Longword-Write Burst to 8-Bit Port 3-1-1-1 (No Wait States)

Figure 17-28 shows a longword read from an 8-bit device with burst inhibited. The transfer results in four individual transfers.

NOTE

There is an extra clock of address setup (AS) for each burst-inhibited transfer between states S0 and S1.
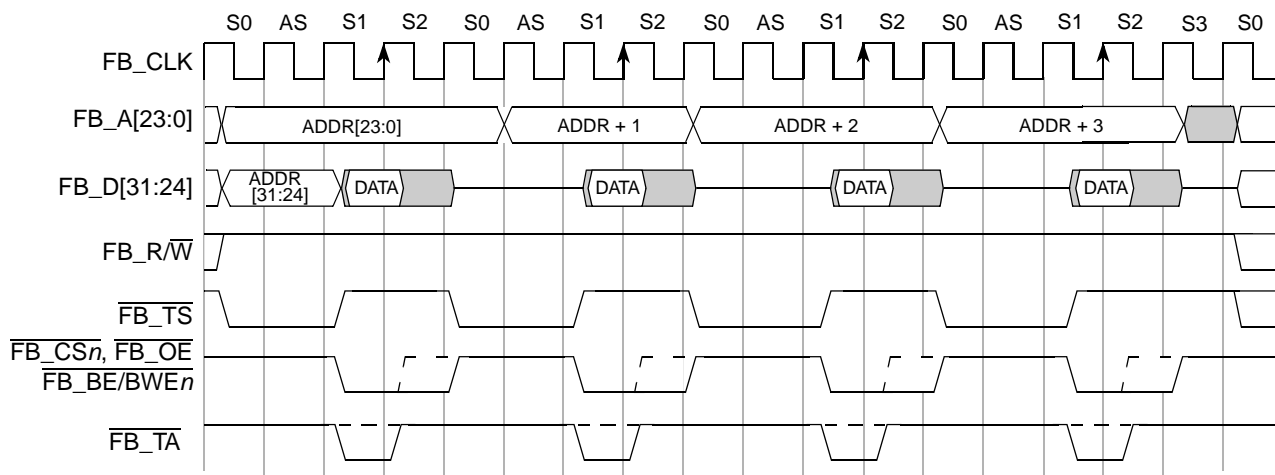
Figure 17-28. Longword-Read Burst-Inhibited from 8-Bit Port (No Wait States)

Figure 17-29 shows a longword write to an 8-bit device with burst inhibited. The transfer results in four individual transfers.
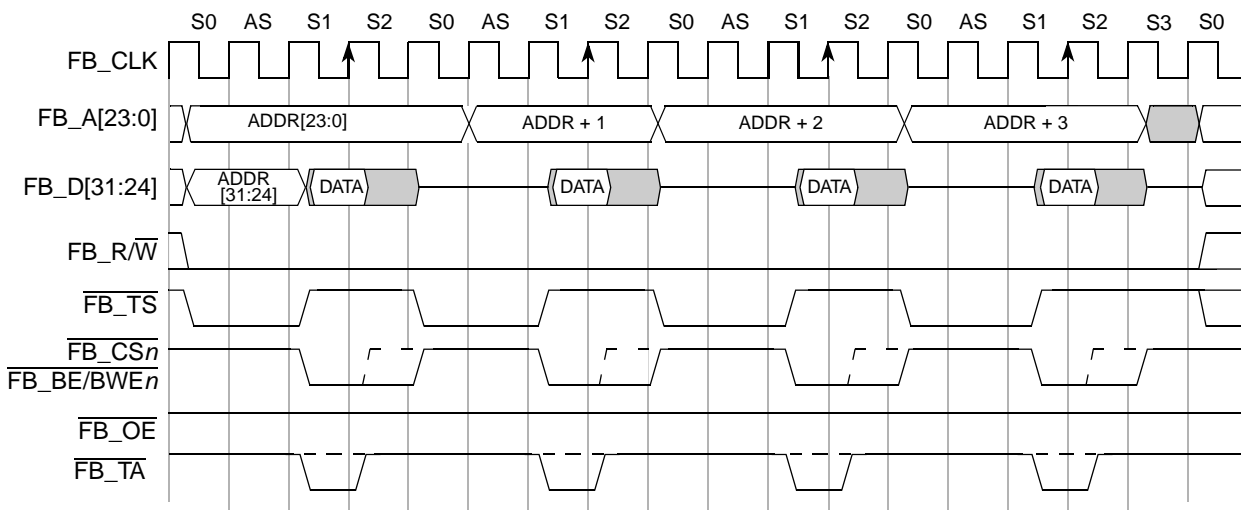
Figure 17-29. Longword-Write Burst-Inhibited to 8-Bit Port (No Wait States)

Figure 17-30 illustrates another read burst transfer, but in this case a wait state is added between individual beats.

NOTE

CSCRn[WS] determines the number of wait states in the first beat. However, for subsequent beats, the CSCRn[WS] (or CSCRn[SWS] if CSCRn[SWSEN] is set) determines the number of wait states.
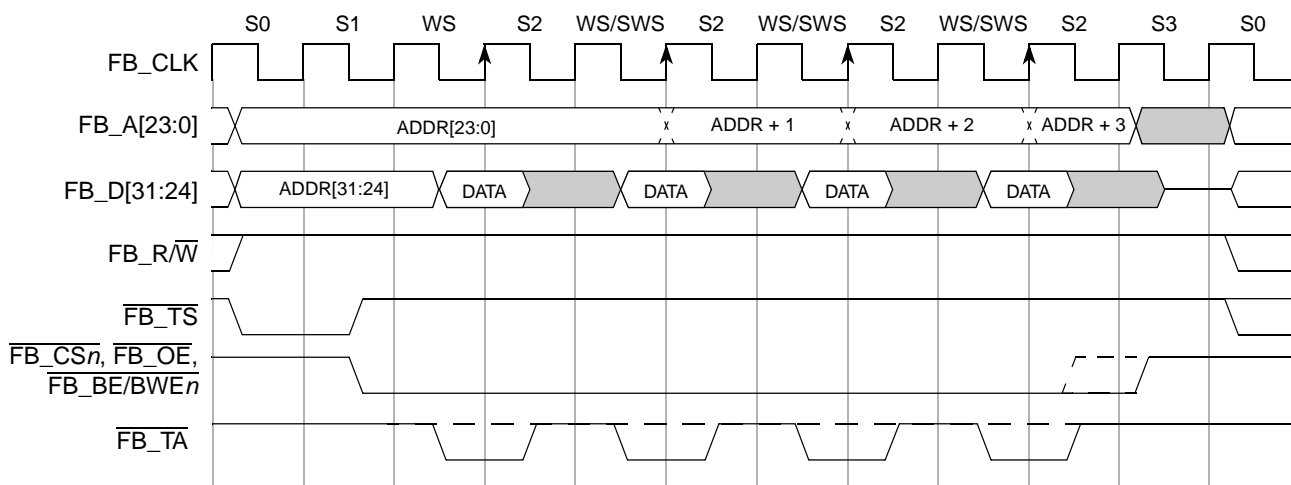
Figure 17-30. Longword-Read Burst from 8-Bit Port 3-2-2-2 (One Wait State)

Figure 17-30 illustrates a write burst transfer with one wait state.
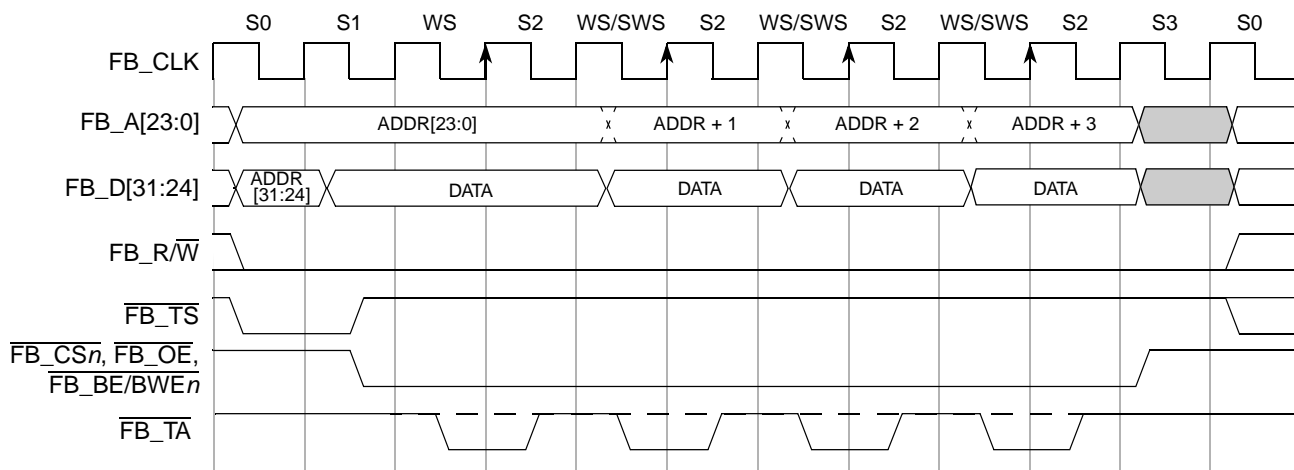
Figure 17-31. Longword-Write Burst to 8-Bit Port 3-2-2-2 (One Wait State)

If address setup and hold are used, only the first and last beat of the burst cycle are affected. Figure 17-32 shows a read cycle with one clock of address setup and address hold.

NOTE

When using internal termination in this scenario (CSCRn[AA] = 1), the address increments after the clock-edge boundary. The attached device must be able to account for this, or a wait state must be added.
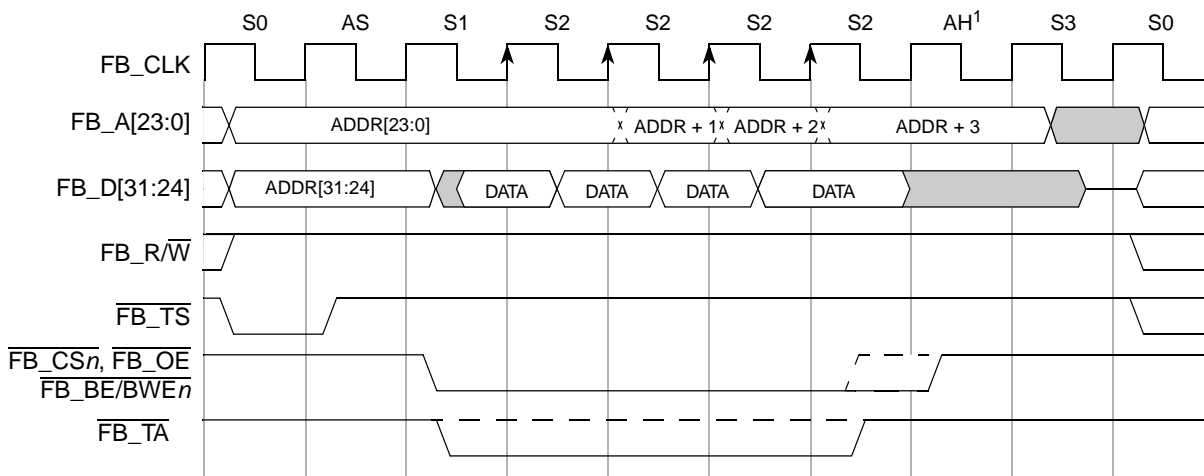
[1] The address hold time depends on the setting of CSCRn[AA]. See Section 17.3.3, "Chip-Select Control Registers (CSCR0 – CSCR5)", for more details.

Figure 17-32. Longword-Read Burst from 8-Bit Port 3-1-1-1 (Address Setup and Hold)

Figure 17-33 shows a write cycle with one clock of address setup and address hold.
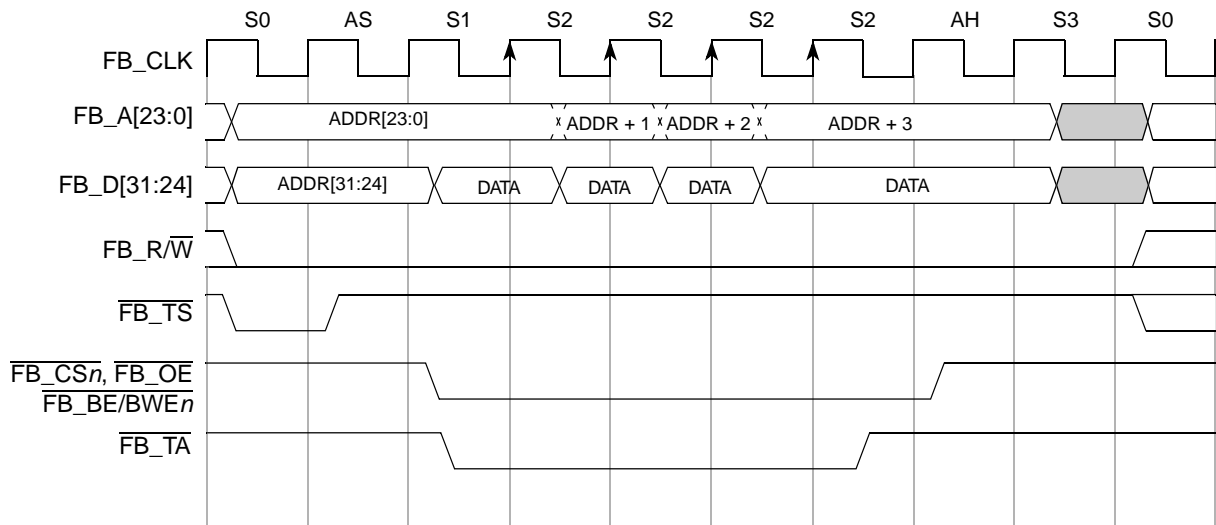
Figure 17-33. Longword-Write Burst to 8-Bit Port 3-1-1-1 (Address Setup and Hold)

17.4.7   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 17-1 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 17-1. 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 17-2 differs from the one in Example 17-1 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 17-2. A Misaligned Word Transfer (32-Bit Port)

17.4.8   Bus Errors

The ColdFire device has no bus monitor. If the auto-acknowledge feature is not enabled for the address that generates the error, the bus cycle can be terminated by asserting $\overline{FB\_TA}$ or by using the software watchdog timer. If the processor must manage a bus error differently, asserting an interrupt to the core along with $\overline{FB\_TA}$ when the bus error occurs can invoke an interrupt handler.

Chapter 18
SDRAM Controller (SDRAMC)

18.1 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 ($f_{sys/3}$).

The external data bus is shared between the FlexBus module and the SDRAM controller. When the SDRAM controller is in SDR mode (DRAMSEL = 1), the data bus is switched dynamically between the SDRAM controller and the FlexBus module. However, when the SDRAM controller is in DDR mode (DRAMSEL = 0), D[31:16] is dedicated to the SDRAM data bus and D[15:0] is dedicated to the FlexBus data bus.

In this chapter, the SDRAM data bus signals are named SD_D[31:0]. However, because these signals share external pins with the FlexBus, the pin names on the device are D[31:0].

18.1.1 Block Diagram

Block diagram of the SDRAM controller:
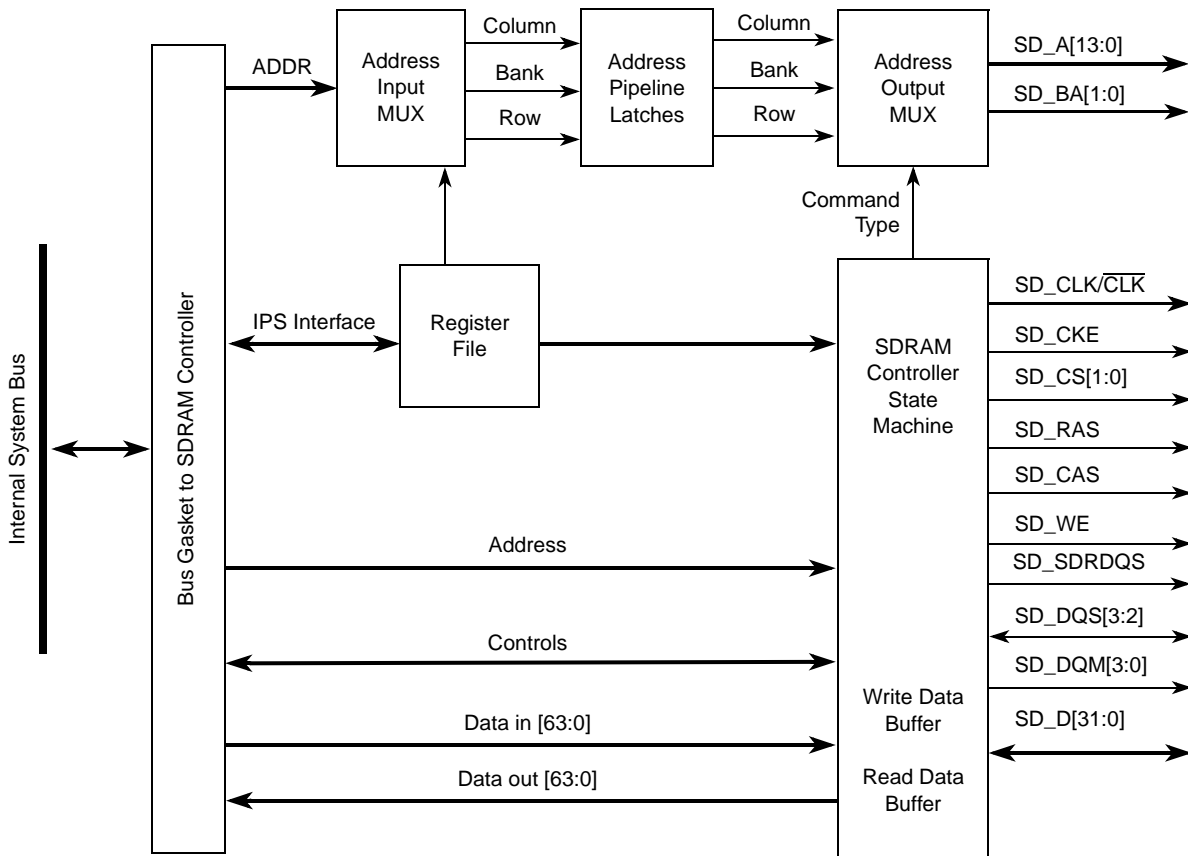
Figure 18-1. SDRAM Controller Block Diagram

18.1.2 Features

The SDRAM controller contains:

- Supports standard SDRAM (single data rate, or SDR) and dual data rate (DDR) SDRAM; one or the other, not mixed.
- Support for lower-power/mobile DDR SDRAM.
- Dynamic 16- or 32-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 12 (in 32-bit bus mode) or 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 24 in 32-bit bus mode or 25 in 16-bit bus mode.
- Minimum memory configuration of 8 MByte
  - 11 bit row address (RA), 8 bit column address (CA), 2 bit bank address (BA), 32-bit bus, one chip select

— 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.

— 24/25 bits RA+CA, 2 bits BA, 32/16-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

18.1.3 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.

18.2 External Signal Description

This section introduces the signal names used in this chapter.

Table 18-1. SDRAM Interface—Detailed Signal Descriptions

Signal	I/O	Description	
SD_A[13:0]	O	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). A12 is used on device densities of 256 Mb and above.	
		Timing	Assertion/Negation — Occurs synchronously with SD_CLK

Table 18-1. SDRAM Interface—Detailed Signal Descriptions (continued)

Signal	I/O	Description
SD_BA[1:0]	O	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.
		Timing — Assertion/Negation — Occurs synchronously with SD_CLK
$\overline{SD\_CAS}$	O	Column address strobe/command input. Along with $\overline{SD\_CS}$, $\overline{SD\_RAS}$, and $\overline{SD\_WE}$, defines the current command.
		State Meaning — See Table 18-12 for the SDRAM commands.
		Timing — Assertion/Negation — Occurs synchronously with SD_CLK
$\overline{SD\_RAS}$	O	Row address strobe/command input. Along with $\overline{SD\_CS}$, $\overline{SD\_CAS}$, and $\overline{SD\_WE}$, defines the current command.
		State Meaning — See Table 18-12 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, $\overline{SD\_CLK}$, 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 $\overline{SD\_CLK}$	O	SD_CLK and $\overline{SD\_CLK}$ 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 $\overline{SD\_CLK}$. Output data is referenced to the crossing of SD_CLK and $\overline{SD\_CLK}$ (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.
$\overline{SD\_CS}$[1:0]	O	$\overline{SD\_CS}$ provides external bank selection on systems with multiple banks. $\overline{SD\_CS}$ 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[31:0]	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 $\overline{SD\_CLK}$. 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.

Table 18-1. SDRAM Interface—Detailed Signal Descriptions (continued)

Signal	I/O	Description
$\overline{\text{SD\_DQM}}$[3:0]	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] SD_DQM1 - SD_D[15:8] SD_DQM0 - SD_D[7:0]
		State Meaning: Asserted — Data is written to SDRAM Negation — Data is masked
		Timing: Assertion/Negation — Occurs on crossing of SD_CLK and $\overline{\text{SD\_CLK}}$.
SD_DQS[3:2]	I/O	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] **Note:** 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.
		State Meaning: 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.
		Timing: Assertion/Negation — Occurs on crossing of SD_CLK and $\overline{\text{SD\_CLK}}$.
SD_SDRDQS	O	SDR data strobe. Generated by the memory controller in SDR mode, to mimic the DQS generated by DDR memories during reads. It should be routed out and connected back to the SD_DQS inputs.
		State Meaning: Asserted— Similar to a clock signal, the edges are more important than being asserted or negated.
		Timing: Assertion/Negation—Occurs on crossing of SD_CLK and $\overline{\text{SD\_CLK}}$.
$\overline{\text{SD\_WE}}$	O	Command input. Along with $\overline{\text{SD\_CS}}$, $\overline{\text{SD\_CAS}}$, and $\overline{\text{SD\_RAS}}$ defines the current command.
		State Meaning: Please see Table 18-12 for SDRAM commands.
		Timing: Assertion/Negation— Occurs synchronously with SD_CLK.

18.3   Interface Recommendations

18.3.1   Supported Memory Configurations

The SDRAM controller supports up to 14 row addresses and up to (13 in 16-bit bus mode) 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 24 (25 in 16-bit bus mode). 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^{26} x 32 bit (2^{27} x 16 bit) or 256 MBytes.

Table 18-5 and Table 18-6 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). In 32-bit bus mode, IA[9:2] are used for CA[7:0]. In 16-bit mode, IA[9:1] are used for CA[8:0]. IA[11:10] are always used for BA[1:0], and IA[23:12] are always 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.

NOTE

When the SDRAMC is configured to support an external 32-bit data bus. It is not possible to connect a smaller device(s) to only part of the SDRAM's data bus. For example, if 16-bit wide devices are used, then user must use two 16-bit devices connected as a 32-bit port.

Table 18-2. Address Multiplexing for 32-bit Bus Mode

SDCR[ADDR_MUX]	Internal Address Bits [27:24]			
	IA[27]	IA[26]	IA[25]	IA[24]
00	CA12	CA11	CA9	CA8
01	CA11	CA9	CA8	RA12
10	CA9	CA8	RA13	RA12
11	Reserved, do not use.			

Table 18-3. SDRAM Address Multiplexing in 32-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	9–2
64 Mbits	2M x 32 bit	11 x 8 x 4	00	—[1,2]	—	—	—	RA11-0	BA1-0	CA7-0
	4M x 16 bit	12 x 8 x 4	00	—	—	—	—			
	8M x 8 bit	12 x 9 x 4	00	—	—	—	CA8			
		13 x 8 x 4	01	—	—	—	RA12			
	16M x 4 bit	12 x 10 x 4	00	—	—	CA9	CA8			
		13 x 9 x 4	01	—	—	CA8	RA12			

Table 18-3. SDRAM Address Multiplexing in 32-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	9–2
128 Mbits	4M x 32 bit	12 x 8 x 4	00	—	—	—	—	RA11-0	BA1-0	CA7-0
	8M x 16 bit	12 x 9 x 4	00	—	—	—	CA8			
		13 x 8 x 4	01	—	—	—	RA12			
	16M x 8 bit	12 x 10 x 4	00	—	—	CA9	CA8			
		13 x 9 x 4	01	—	—	CA8	RA12			
		14 x 8 x 4	10	—	—	RA13	RA12			
	32M x 4 bit	12 x 11 x 4	00	—	CA11	CA9	CA8			
		13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			
256 Mbits	8M x 32 bit	12 x 9 x 4	00	—	—	—	CA8	RA11-0	BA1-0	CA7-0
		13 x 8 x 4	01	—	—	—	RA12			
	16M x 16 bit	12 x 10 x 4	00	—	—	CA9	CA8			
		13 x 9 x 4	01	—	—	CA8	RA12			
		14 x 8 x 4	10	—	—	RA13	RA12			
	32M x 8 bit	12 x 11 x 4	00	—	CA11	CA9	CA8			
		13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			
	64M x 4 bit	12 x 12 x 4	00	CA12	CA11	CA9	CA8			
		13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			
512 Mbits	16M x 32 bit	12 x 10 x 4	00	—	—	CA9	CA8	RA11-0	BA1-0	CA7-0
		13 x 9 x 4	01	—	—	CA8	RA12			
		14 x 8 x 4	10	—	—	RA13	RA12			
	32 M x 16 bit	12 x 11 x 4	00	—	CA11	CA9	CA8			
		13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			
	64M x 8 bit	12 x 12 x 4	00	CA12	CA11	CA9	CA8			
		13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			

Table 18-3. SDRAM Address Multiplexing in 32-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	9–2
1 Gbits	32M x 32 bit	12 x 11 x 4	00	—	CA11	CA9	CA8	RA11-0	BA1-0	CA7-0
		13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			
	64M x 16 bit	12 x 12 x 4	00	CA12	CA11	CA9	CA8			
		13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			
2 Gbits	64M x 32 bit	12 x 12 x 4	00	CA12	CA11	CA9	CA8	RA11-0	BA1-0	CA7-0
		13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	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.

Table 18-4. 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 18-5. 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			

Table 18-5. 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
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			
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.

For 32-bit data bus configuration:

- If all devices' row address width is 12 bits, the column address can be ≥ 8 bits.
- If all devices' row address width is 13 bits, the column address can be ≥ 8 bits.
- If all devices' column address width is 8 bits, the row address can be ≥ 11 bits.
- The maximum row bits plus column bits equals 24.
- x8 and x16 data width memory devices can be mixed (but not in the same space).
- x32 data width memory devices cannot be mixed with any other width.

For 16-bit data bus configuration:

- 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.

18.3.2   SDRAM SDR Connections

Figure 18-2 shows a block diagram using 32-bit wide SDR SDRAM (such as Micron MT48LC4M32B2) and flash (such as Spansion AM29LV160D). SDR design requires special timing consideration for the SD_DQS[3:2] signals. For reads from DDR SDRAMs, the memory drives the DQS pins so that the data lines and DQS signals have concurrent edges. The SDRAMC is designed to latch data 1/4 clock after the SD_DQS[3:2] edge. For DDR SDRAM, this ensures that the latch time is in the middle of the data valid window.

The SDRAMC also uses the SD_DQS[3:2] signals to determine when read data can be latched for SDR SDRAM; however, SDR memories do not provide DQS outputs. Instead the SDRAMC provides a SD_SDRDQS output routed back into the controller as SD_DQS[3:2]. The SD_SDRDQS signal should be routed such that the valid data from the SDRAM reaches the controller at the same time or before the SD_SDRDQS reaches the SD_DQS[3:2] inputs.

When routing SD_SDRDQS the outbound trace length should be matched to the SD_CLK trace length. This aligns SD_SDRDQS to the SD_CLK as if the memory had generated the DQS pulse. The inbound trace should be routed along the data path, which should synchronize the SD_DQS so that the data is latched in the middle of the data valid window.
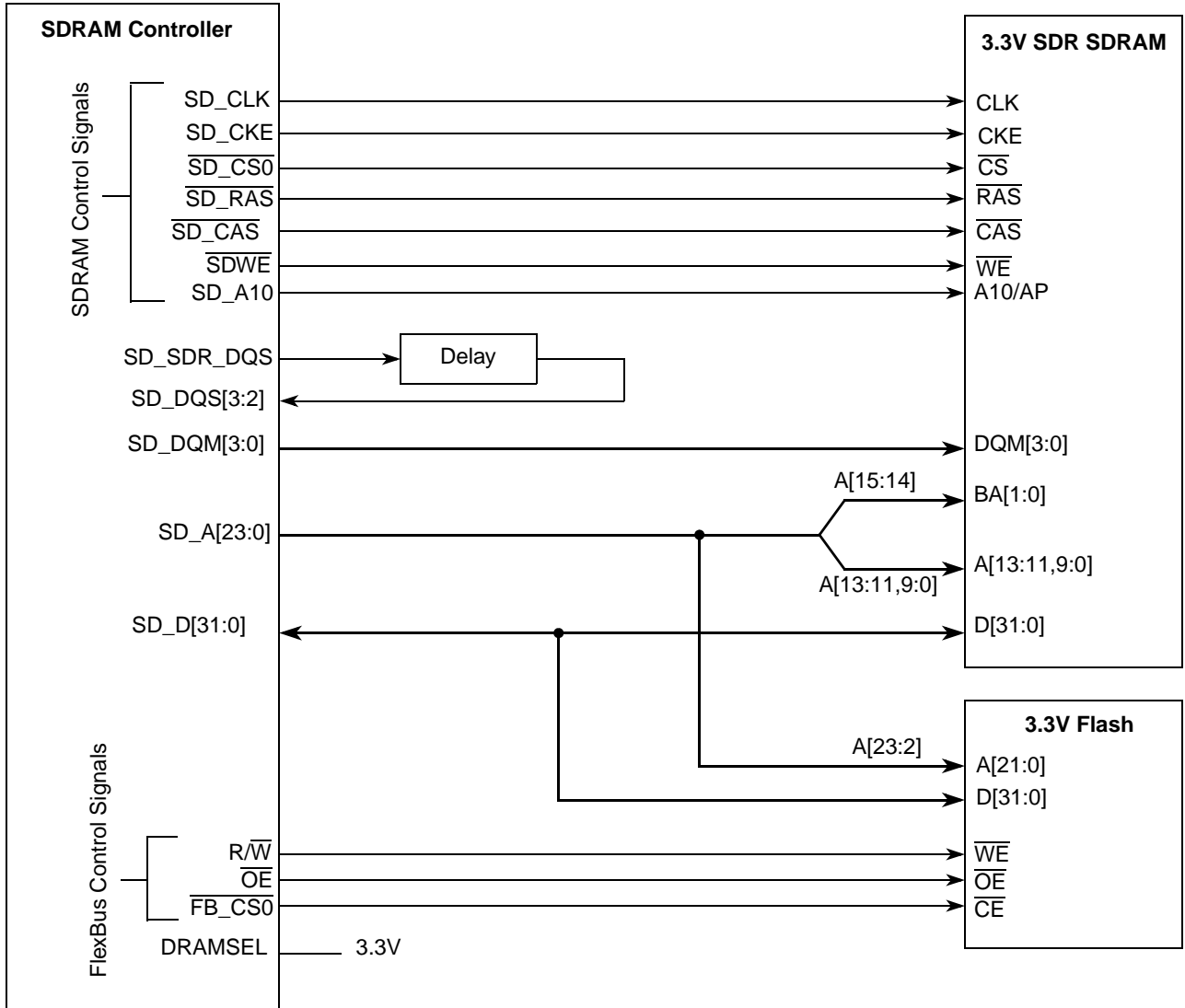
Figure 18-2. Example 3.3V, 32-bit SDR SDRAM System

18.3.3 SDRAM DDR Component Connections

Figure 18-3 shows a block diagram using 16-bit wide DDR SDRAM (such as Micron MT46V8M16) and flash (such as Spansion AM29DBB160G).
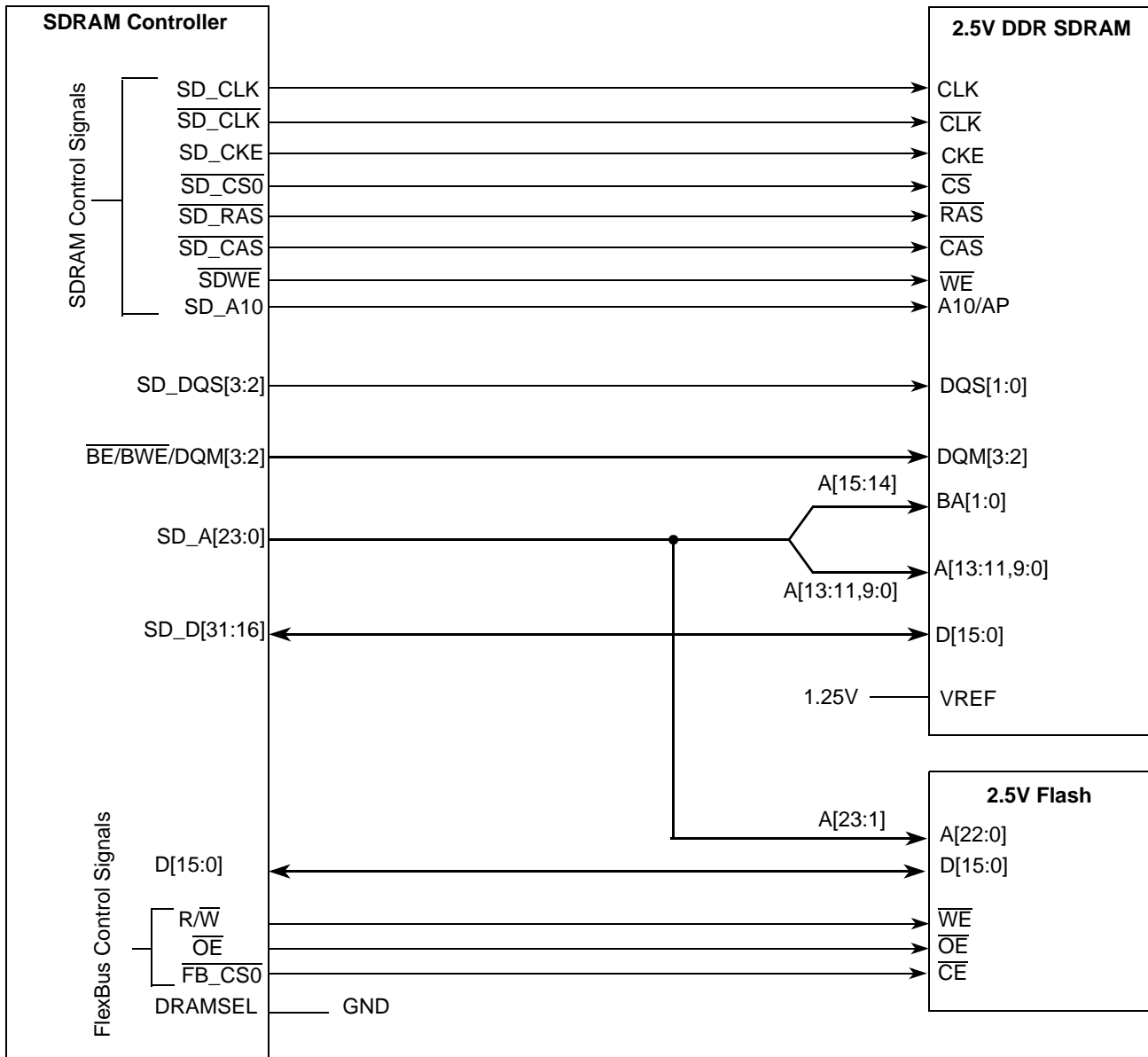
Figure 18-3. Example 2.5V, 16-bit DDR SDRAM System

18.3.4 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.
- The loading and routing of SD_DQS must match those of SD_D.
- 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.
- Series termination should be used to help match output driver impedance to trace impedance. (Driver impedance is affected by drive strength.) Typically, a 50 Ω system with a 22 Ω series resistor is a good starting point, but this should be analyzed based on actual board design and loading.
- 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.

NOTE

Only series termination is supported on this device, which is different than typical DDR designs.

18.3.4.1 Termination Example

Figure 18-4 shows the recommended termination circuitry for DDR SDRAM signals.
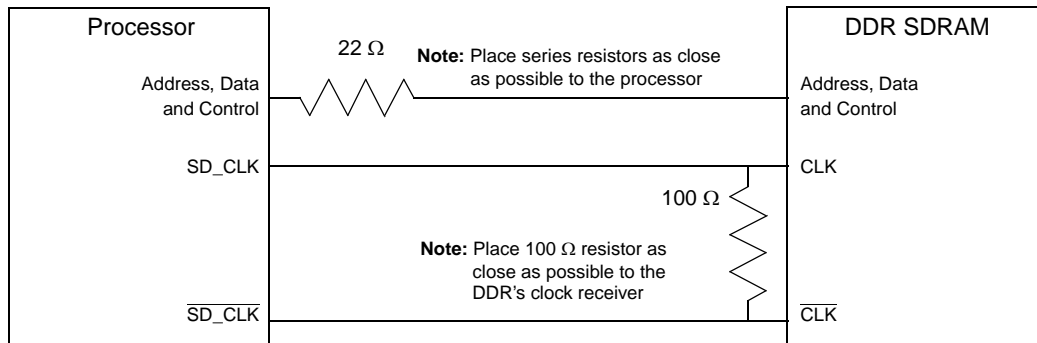
Figure 18-4. DDR SDRAM Termination Circuit

18.4 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 GPIO module. See Section 13.3.7, "SDRAM Mode Select Control Register (MSCR_SDRAM)," for more details.

Table 18-6 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 18-6. 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	18.4.1/18-14
0xFC0B_8004	SDRAM Control Register (SDCR)	32	R/W	0x0000_0000	18.4.2/18-15
0xFC0B_8008	SDRAM Configuration Register 1 (SDCFG1)	32	R/W	0x0000_0000	18.4.3/18-17
0xFC0B_800C	SDRAM Configuration Register 2 (SDCFG2)	32	R/W	0x0000_0000	18.4.4/18-19
0xFC0B_8110	SDRAM Chip Select 0 Configuration (SDCS0)	32	R/W	0x0000_0000	18.4.5/18-20
0xFC0B_8114	SDRAM Chip Select 1 Configuration (SDCS1)	32	R/W	0x0000_0000	18.4.5/18-20

18.4.1 SDRAM Mode/Extended Mode Register (SDMR)

The SDMR (Figure 18-5) writes to the mode and extended mode registers physically residing within the SDRAM chips. These registers must be programmed during SDRAM initialization. See Section 18.6, "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
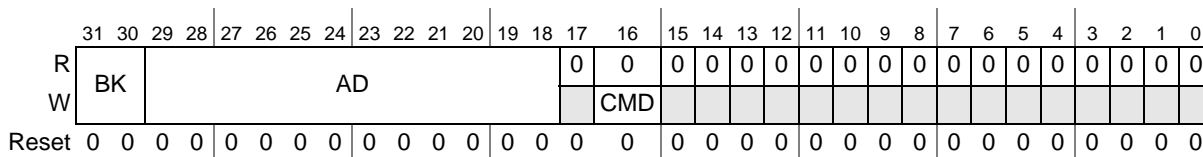
Figure 18-5. SDRAM-Mode/Extended-Mode Register (SDMR)

Table 18-7. 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
15–0	Reserved, must be cleared.

18.4.2 SDRAM Control Register (SDCR)

The SDCR (Figure 18-6) controls SDRAMC operating modes, including refresh count and address line muxing.

Address: 0xFC0B_8004 (SDCR) Access: User read/write

Figure 18-6. SDRAM Control Register (SDCR)

Table 18-8. 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)

Table 18-8. SDCR Field Descriptions (continued)

Field	Description
29 DDR_MODE	DDR mode select. 0 SDR mode 1 DDR mode
28 REF_EN	Refresh enable. 0 Automatic refresh disabled 1 Automatic refresh enabled
27–26	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 18-4, and Table 18-5.
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.
21–16 REF_CNT	The average periodic interval at which the controller generates refresh commands to memory; measured in increments of $64 \times$ SD_CLK period. REF_CNT = (t_{REFI}/ ($t_{CK} \times 64$)) - 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 32-bit data bus 1 16-bit data bus
12	Reserved, must be cleared.
11–10 DQS_OE	DQS output enable. Each DQS_OE bit 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 SDR mode or 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 $\overline{SD\_CSn}$ 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.

Table 18-8. SDCR Field Descriptions (continued)

Field	Description
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 $\overline{SD\_CSn}$ 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	Reserved, should be cleared.

18.4.3   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).

The minimum values of certain fields can be different for SDR, DDR SDRAM, even if the data sheet timing is the same, because:

- In SDR mode, the memory controller counts the delay in SD_CLK
- In DDR mode, the memory controller counts the delay in 2 x SD_CLK (also referred to as 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 18-7. SDRAM Configuration Register 1 (SDCFG1)

Table 18-9. SDCFG1 Field Descriptions

Field	Description
31–28 SRD2RWP	Single read to read/write/precharge delay. Limiting case is read to write. SDR: SRD2RWP = CL + t_{HZ} + 2 DDR: SRD2RWP = CL + 1 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 SDR and DDR mode.
27	Reserved, must be cleared.
26–24 SWT2RWP	Single write to read/write/precharge delay. Limiting case is write to precharge. SDR: SWT2RWP = t_{WR} DDR: SWT2RWP = t_{WR} + 1 **Note:** Count value is in SD_CLK periods for SDR and DDR mode.
23–20 RD_LAT	Read CAS Latency. Read command to read data available delay counter. For DDR: 　　If CL = 2, write 0x6 　　If CL = 2.5, write 0x7 For SDR: 　　If CL = 2, write 0x2 　　If CL = 3, write 0x3 **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. 　　CL = 2.5 is not supported for SDR. 　　SDR: Count value is in SD_CLK periods. 　　DDR: Count value is in SD_CLK2 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 $\times$ 99 MHz) - 1= 0.98; round to 1. **Note:** Count value is in SD_CLK periods for SDR and DDR modes.

Table 18-9. SDCFG1 Field Descriptions (continued)

Field	Description
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 $\times$ 99 MHz) - 1 = 0.98; round to 1.
	
	Note: Count value is in SD_CLK periods for SDR and DDR modes.

11–8 REF2ACT	Refresh to active delay. Refresh command to following active or refresh command delay counter.
	
	SDR/DDR: REF2ACT = $(t_{RFC} \times f_{SD\_CLK})$ - 1 (Round up to nearest integer)
	Example (for SDR/DDR):
	    If t_{RFC} = 75ns and f_{SD_CLK} = 99 MHz
	    Suggested value = (75ns $\times$ 99 MHz) - 1 = 6.425; round to 7.
	
	Note: Count value is in SD_CLK periods for SDR and DDR modes.

7	Reserved, must be cleared.
6–4 WT_LAT	Write latency. Write command to write data delay counter. SDR: write 0x0 DDR: write 0x3 **Note:** SDR mode: Count value is in SD_CLK periods. DDR mode: Count value is in SD_CLK2 periods.
3–0	Reserved, must be cleared.

18.4.4 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
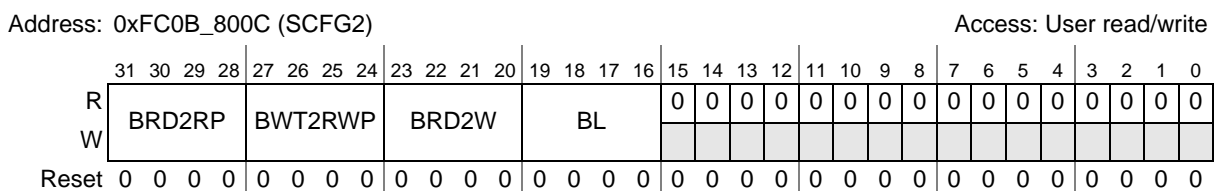
Figure 18-8. SDRAM Configuration Register 2 (SDCFG2)

Table 18-10. SDCFG2 Field Descriptions

Field	Description
31–28 BRD2RP	Burst read to read/precharge delay. Limiting case is read to read. SDR: BRD2RP = BurstLength + 1 DDR: BRD2RP = BurstLength/2 + 1
27–24 BWT2RWP	Burst write to read/write/precharge delay. Limiting case is write to precharge. SDR: BWT2RWP = BurstLength + t_{WR} - 2 DDR: BWT2RWP = BurstLength/2 + t_{WR}
23–20 BRD2W	Burst read to write delay. SDR: BRD2W = CL + BurstLength + t_{HZ} DDR: BRD2W = CL + BurstLength/2 - 1
19–16 BL	Burst length. BL = BurstLength - 1 **Note:** Burst length depends on port sizeIf 32-bit bus (SDCR[MEM_PS] = 0), burst length is 4. Write BL = 3.. If 16-bit bus (SDCR[MEM_PS] = 1), burst length is 8. Write BL = 7.
15–0	Reserved, must be cleared.

18.4.5 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)   Access: User read/write
          0xFC0B_8114 (SDCS1)

	31	30	29	28	27	26	25	24	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
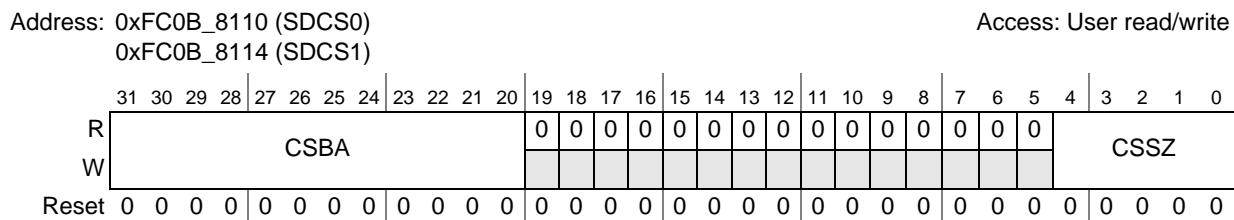
Figure 18-9. SRAM Chip Select Configuration Register (SDCS*n*)

Table 18-11. 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.
19–5	Reserved, must be cleared.
4–0 CSSZ	Chip select size.

CSSZ	Size	Address Lines to Compare	CSSZ	Size	Address Lines to Compare
00000	Disabled	—	11000	32 MByte	A[31:25]
00001–10001	Reserved	Reserved	11001	64 MByte	A[31:26]
10010	Reserved	Reserved	11010	128 MByte	A[31:27]
10011	1 MByte	A[31:20]	11011	256 MByte	A[31:28]
10100	2 MByte	A[31:21]	11100	512 MByte	A[31:29]
10101	4 MByte	A[31:22]	11101	1 GByte	A[31:30]
10110	8 MByte	A[31:23]	11110	2 GByte	A31
10111	16 MByte	A[31:24]	11111	4 GByte	Ignore A[31:20]

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.

18.5 Functional Description

18.5.1 SDRAM Commands

When an internal bus master accesses SDRAM address space, the memory controller generates the corresponding SDRAM command. Table 18-12 lists SDRAM commands supported by the memory controller.

Table 18-12. SDRAM Commands

Function	Symbol	CKE	$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{WE}$	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
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 (SDR/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
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).

18.5.1.1 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 18.6.1, "Page Management," for more information.

18.5.1.2 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 SDR memory, the data masks are negated throughout the entire read size. With DDR memory, the data masks are asserted high throughout the entire read size; but DDR memory ignores the data masks during reads.

18.5.1.3　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.

In SDR mode, the memory controller issues the burst terminate command to truncate burst write for a single write. This is not the case for DDR system. With SDR and 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.

18.5.1.4　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 (SDR and DDR) and write bursts (SDR). To truncate a burst write for DDR, the read command can abort the remaining unnecessary write beats. This method also works when in SDR mode. The most recently registered read or write command prior to the burst terminate command is truncated. The active page remains open.

18.5.1.5　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.

18.5.1.6 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, step 2 to 4 should be performed twice. The first is for the extended-mode register, and the last is for the mode register.
6. Clear the SDCR[MODE_EN] bit.

18.5.1.6.1 Mode Register Definition

Figure 18-10 shows the mode register definition. This is the SDRAM's mode register, not the SDRAMC's mode/extended mode register (SDMR) defined in Section 18.4.1, "SDRAM Mode/Extended Mode Register (SDMR)." Refer to device data sheet for detailed description.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	0			OP_MODE				CL		BT		BLEN	
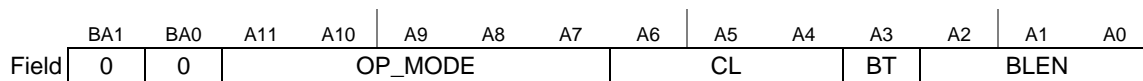
Figure 18-10. Mode Register

Table 18-13. 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. *xx*000 Standard Operation (SDR only) 00000 Normal Operation (DDR) 00010 Reset DLL (DDR) 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 18-13. 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. 000 One. This setting is not valid for DDR. 001 Two 010 Four 011 Eight Else Reserved

18.5.1.6.2 Extended Mode Register Definition

Figure 18-11 shows the 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 18.4.1, "SDRAM Mode/Extended Mode Register (SDMR)." Refer to device data sheet for detailed description.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	1						OPTION						DLL
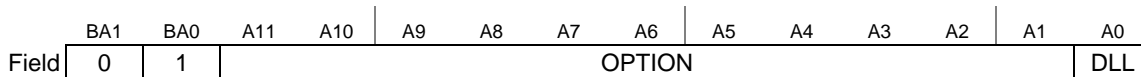
Figure 18-11. Extended Mode Register

Table 18-14. 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

18.5.1.7 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.

18.5.1.8 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.

18.5.2 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 18-12 displays a simple timing diagram that illustrates the end result of the RCR delay.
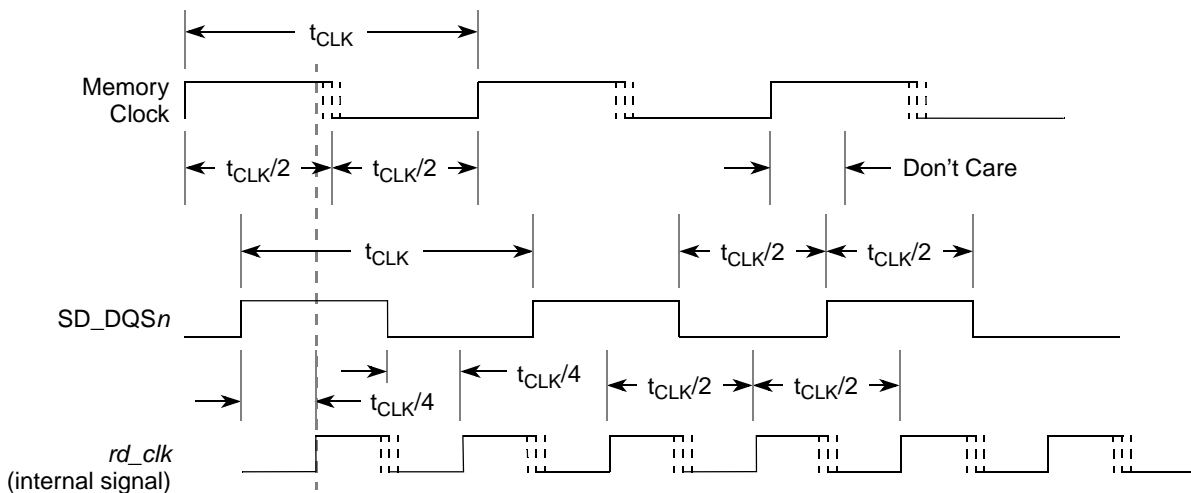
Figure 18-12. Frequency Doubler Block Diagram

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.

Single data rate (SDR) memories do not use or provide DQS signals. In SDR mode, the SDRAM controller provides an SDRDQS signal that can be driven off-chip and routed back into the DQS inputs. The center of the data valid window is guaranteed relative to the memory clock at the memory devices.

The round-trip SDRDQS-to-DQS delay must match the memory clock output + read data input + round-trip time-of-flight delay so that the received DQS edges have a known phase relationship to the received data. The SDRDQS signal is generated with a 1/4 memory clock period lead time, to compensate for the 1/4 memory clock period delay of DQS through the RCR delay module.

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.

18.6 Initialization/Application Information

SDRAMs have a prescribed initialization sequence. The following section details the memory initialization steps for DDR SDRAM. The sequence might change slightly from device-to-device. Refer to the device datasheet as the most relevant reference.

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 $\overline{SD\_CS}$ pins in GPIO 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] 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 18.5.1.6, "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 18.5.1.6, "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] set. The SDCR[REF and IREF] bits should remain cleared for this step.

11. Refresh the SDRAM. The SDRAM specification should indicate many refresh cycles performed before issuing an LMR command. Write to the SDCR with the IREF bit 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 18.5.1.6, "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.

18.6.1   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 $\overline{SD\_CS}$ 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 $\overline{SD\_CS}$ 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 $\overline{SD\_CS}$ 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 $\overline{SD\_CS}$ blocks are refreshed at the same time. The refresh closes all banks of every SDRAM block.

18.6.2 Transfer Size

In the SDRAMC, the internal data bus is 32 bits wide, while the SDRAM external interface bus is 32 or 16 bits wide. Therefore, each internal data beat requires one or 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, four beats of 4 bytes (32-bit mode), 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 19
Fast Ethernet Controller (FEC)

19.1 Introduction

This chapter provides a feature-set overview, a functional block diagram, and transceiver connection information for the 10 and 100 Mbps MII (media independent interface), as well as the 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

19.1.1 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 FEC supports three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbps MII and the 10 Mbps-only 7-wire interface.

NOTE

The module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 13, "General Purpose I/O Module") prior to configuring the FEC.

19.1.2 Block Diagram

Figure 19-1 shows the block diagram of the FEC. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.
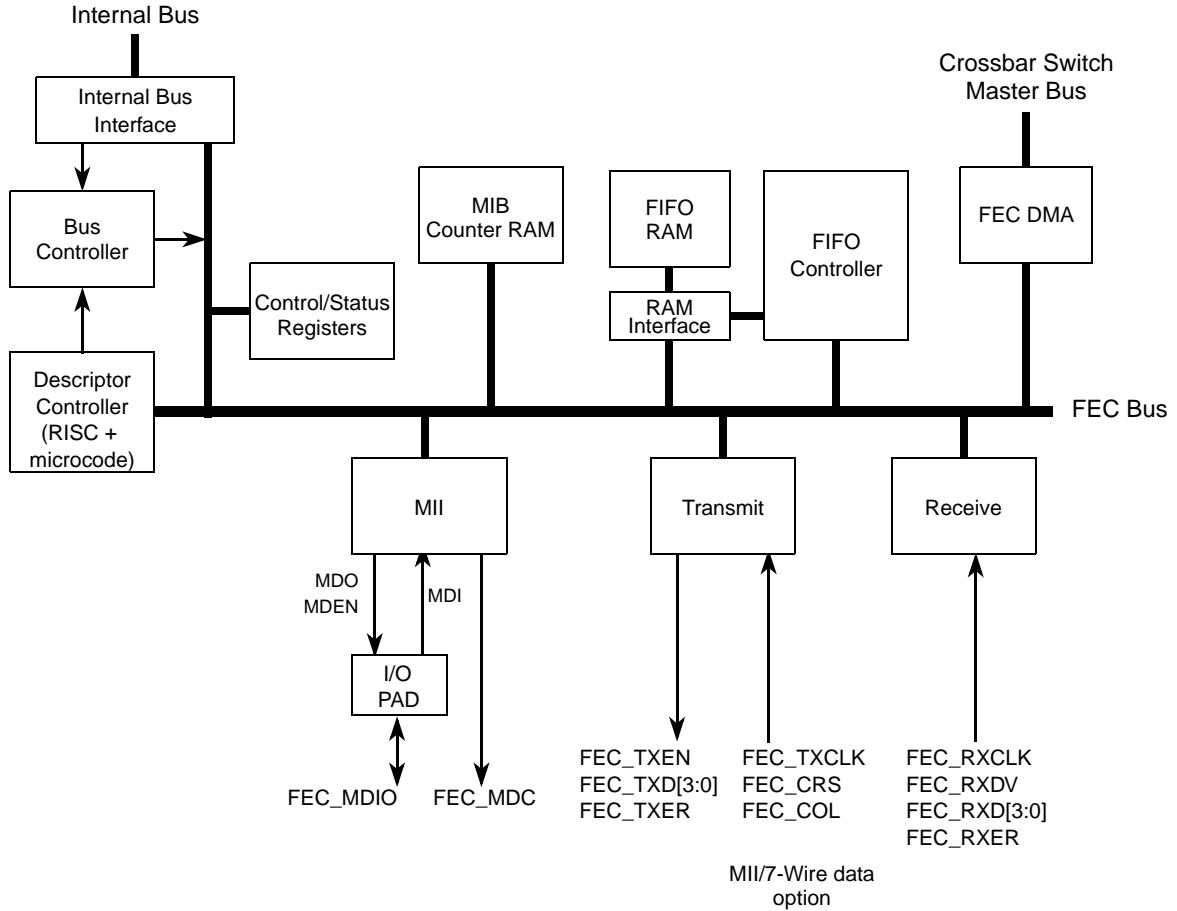
Figure 19-1. FEC Block Diagram

The descriptor controller is a RISC-based controller providing these functions in the FEC:

- 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 16, "Enhanced Direct Memory Access (eDMA)," nor to the DMA timers described in Chapter 29, "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 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 FEC 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_MDC (management data clock) and FEC_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 19.4.1, "MIB Block Counters Memory Map," for more information.

19.1.3   Features

The FEC incorporates the following features:

- Support for three different Ethernet physical interfaces:
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 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

19.2 Modes of Operation

The primary operational modes are described in this section.

19.2.1 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[FDEN] controls duplex mode selection.

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC_PAUSE,TFC_PAUSE] bits, the RCR[FCE] bit, and Section 19.5.11, "Full Duplex Flow Control," for more details.

19.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in Section 19.5.6, "Network Interface Options."

19.2.2.1 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[MII_MODE].

FEC_TXCLK and FEC_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_MDC/FEC_MDIO pins) to the transceiver. Refer to the MMFR and MSCR 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.

19.2.2.2 10 Mpbs 7-Wire Interface Operation

The FEC supports 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.

19.2.3　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 19.5.9, "Ethernet Address Recognition."

19.2.4　Internal Loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in Section 19.5.14, "MII Internal and External Loopback."

19.3　External Signal Description

Table 19-1 describes the various FEC signals, as well as indicating which signals work in available modes.

Table 19-1. FEC Signal Descriptions

Signal Name	MII	7-wire	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	—	When asserted, indicates that transmit or receive medium is not idle.
FEC_MDC	X	—	Output clock which 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 is operated 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	Asserting the FEC_RXDV input indicates that the PHY has valid nibbles present on the MII. FEC_RXDV should 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.
FEC_RXD0	X	X	This pin contains the Ethernet input data transferred from the PHY to the media-access controller when FEC_RXDV is asserted.
FEC_RXD1	X	—	This pin contains the Ethernet input data transferred from the 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 the PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXER	X	—	When asserted with FEC_RXDV, indicates that the PHY has detected an error in the current frame. When FEC_RXDV is not asserted FEC_RXER has no effect.
FEC_TXCLK	X	X	Input clock which provides a timing reference for FEC_TXEN, FEC_TXD[3:0] and FEC_TXER.
FEC_TXD0	X	X	The serial output Ethernet data and is only valid during the assertion of FEC_TXEN.
FEC_TXD1	X	—	This pin contains the serial output Ethernet data and is valid only during assertion of FEC_TXEN.
FEC_TXD[3:2]	X	—	These pins contain the serial output Ethernet data and are valid only during assertion of FEC_TXEN.

Table 19-1. FEC Signal Descriptions (continued)

Signal Name	MII	7-wire	Description
FEC_TXEN	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, the PHY sends one or more illegal symbols. FEC_TXER has no effect at 10 Mbps or when FEC_TXEN is negated.

19.4 Memory Map/Register Definition

The FEC is 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 19-2 defines the top level memory map.

Table 19-2. Module Memory Map

Address	Function
0xFC03_0000 – FC03_01FF	Control/Status Registers
0xFC03_0200 – FC03_02FF	MIB Block Counters

Table 19-3 shows the FEC register memory map.

Table 19-3. FEC Register Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_0004	Interrupt Event Register (EIR)	32	R/W	0x0000_0000	19.4.2/19-9
0xFC03_0008	Interrupt Mask Register (EIMR)	32	R/W	0x0000_0000	19.4.3/19-11
0xFC03_0010	Receive Descriptor Active Register (RDAR)	32	R/W	0x0000_0000	19.4.4/19-11
0xFC03_0014	Transmit Descriptor Active Register (TDAR)	32	R/W	0x0000_0000	19.4.5/19-12
0xFC03_0024	Ethernet Control Register (ECR)	32	R/W	0xF000_0000	19.4.6/19-13
0xFC03_0040	MII Management Frame Register (MMFR)	32	R/W	Undefined	19.4.7/19-13
0xFC03_0044	MII Speed Control Register (MSCR)	32	R/W	0x0000_0000	19.4.8/19-15
0xFC03_0064	MIB Control/Status Register (MIBC)	32	R/W	0x0000_0000	19.4.9/19-16
0xFC03_0084	Receive Control Register (RCR)	32	R/W	0x05EE_0001	19.4.10/19-16
0xFC03_00C4	Transmit Control Register (TCR)	32	R/W	0x0000_0000	19.4.11/19-17

Table 19-3. FEC Register Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_00E4	Physical Address Low Register (PALR)	32	R/W	Undefined	19.4.12/19-18
0xFC03_00E8	Physical Address High Register (PAUR)	32	R/W	See Section	19.4.13/19-19
0xFC03_00EC	Opcode/Pause Duration (OPD)	32	R/W	See Section	19.4.14/19-19
0xFC03_0118	Descriptor Individual Upper Address Register (IAUR)	32	R/W	Undefined	19.4.15/19-20
0xFC03_011C	Descriptor Individual Lower Address Register (IALR)	32	R/W	Undefined	19.4.16/19-20
0xFC03_0120	Descriptor Group Upper Address Register (GAUR)	32	R/W	Undefined	19.4.17/19-21
0xFC03_0124	Descriptor Group Lower Address Register (GALR)	32	R/W	Undefined	19.4.18/19-21
0xFC03_0144	Transmit FIFO Watermark (TFWR)	32	R/W	0x0000_0000	19.4.19/19-21
0xFC03_014C	FIFO Receive Bound Register (FRBR)	32	R	0x0000_0600	19.4.20/19-22
0xFC03_0150	FIFO Receive FIFO Start Register (FRSR)	32	R	0x0000_0500	19.4.21/19-22
0xFC03_0180	Pointer to Receive Descriptor Ring (ERDSR)	32	R/W	Undefined	19.4.22/19-23
0xFC03_0184	Pointer to Transmit Descriptor Ring (ETDSR)	32	R/W	Undefined	19.4.23/19-23
0xFC03_0188	Maximum Receive Buffer Size (EMRBR)	32	R/W	Undefined	19.4.24/19-24

19.4.1 MIB Block Counters Memory Map

The MIB counters memory map (Table 19-4) 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 19-4. MIB Counters Memory Map

Address	Register
0xFC03_0200	Count of frames not counted correctly (RMON_T_DROP)
0xFC03_0204	RMON Tx packet count (RMON_T_PACKETS)
0xFC03_0208	RMON Tx broadcast packets (RMON_T_BC_PKT)
0xFC03_020C	RMON Tx multicast packets (RMON_T_MC_PKT)

Table 19-4. MIB Counters Memory Map (continued)

Address	Register
0xFC03_0210	RMON Tx packets with CRC/align error (RMON_T_CRC_ALIGN)
0xFC03_0214	RMON Tx packets < 64 bytes, good CRC (RMON_T_UNDERSIZE)
0xFC03_0218	RMON Tx packets > MAX_FL bytes, good CRC (RMON_T_OVERSIZE)
0xFC03_021C	RMON Tx packets < 64 bytes, bad CRC (RMON_T_FRAG)
0xFC03_0220	RMON Tx packets > MAX_FL bytes, bad CRC (RMON_T_JAB)
0xFC03_0224	RMON Tx collision count (RMON_T_COL)
0xFC03_0228	RMON Tx 64 byte packets (RMON_T_P64)
0xFC03_022C	RMON Tx 65 to 127 byte packets (RMON_T_P65TO127)
0xFC03_0230	RMON Tx 128 to 255 byte packets (RMON_T_P128TO255)
0xFC03_0234	RMON Tx 256 to 511 byte packets (RMON_T_P256TO511)
0xFC03_0238	RMON Tx 512 to 1023 byte packets (RMON_T_P512TO1023)
0xFC03_023C	RMON Tx 1024 to 2047 byte packets (RMON_T_P1024TO2047)
0xFC03_0240	RMON Tx packets with > 2048 bytes (RMON_T_P_GTE2048)
0xFC03_0244	RMON Tx Octets (RMON_T_OCTETS)
0xFC03_0248	Count of transmitted frames not counted correctly (IEEE_T_DROP)
0xFC03_024C	Frames transmitted OK (IEEE_T_FRAME_OK)
0xFC03_0250	Frames transmitted with single collision (IEEE_T_1COL)
0xFC03_0254	Frames transmitted with multiple collisions (IEEE_T_MCOL)
0xFC03_0258	Frames transmitted after deferral delay (IEEE_T_DEF)
0xFC03_025C	Frames transmitted with late collision (IEEE_T_LCOL)
0xFC03_0260	Frames transmitted with excessive collisions (IEEE_T_EXCOL)
0xFC03_0264	Frames transmitted with Tx FIFO underrun (IEEE_T_MACERR)
0xFC03_0268	Frames transmitted with carrier sense error (IEEE_T_CSERR)
0xFC03_026C	Frames transmitted with SQE error (IEEE_T_SQE)
0xFC03_0270	Flow control pause frames transmitted (IEEE_T_FDXFC)
0xFC03_0274	Octet count for frames transmitted without error (IEEE_T_OCTETS_OK)
0xFC03_0280	Count of received frames not counted correctly (RMON_R_DROP)
0xFC03_0284	RMON Rx packet count (RMON_R_PACKETS)
0xFC03_0288	RMON Rx broadcast packets (RMON_R_BC_PKT)
0xFC03_028C	RMON Rx multicast packets (RMON_R_MC_PKT)
0xFC03_0290	RMON Rx packets with CRC/Align error (RMON_R_CRC_ALIGN)
0xFC03_0294	RMON Rx packets < 64 bytes, good CRC (RMON_R_UNDERSIZE)
0xFC03_0298	RMON Rx packets > MAX_FL bytes, good CRC (RMON_R_OVERSIZE)

Table 19-4. MIB Counters Memory Map (continued)

Address	Register
0xFC03_029C	RMON Rx packets < 64 bytes, bad CRC (RMON_R_FRAG)
0xFC03_02A0	RMON Rx packets > MAX_FL bytes, bad CRC (RMON_R_JAB)
0xFC03_02A4	Reserved (RMON_R_RESVD_0)
0xFC03_02A8	RMON Rx 64 byte packets (RMON_R_P64)
0xFC03_02AC	RMON Rx 65 to 127 byte packets (RMON_R_P65TO127)
0xFC03_02B0	RMON Rx 128 to 255 byte packets (RMON_R_P128TO255)
0xFC03_02B4	RMON Rx 256 to 511 byte packets (RMON_R_P256TO511)
0xFC03_02B8	RMON Rx 512 to 1023 byte packets (RMON_R_P512TO1023)
0xFC03_02BC	RMON Rx 1024 to 2047 byte packets (RMON_R_P1024TO2047)
0xFC03_02C0	RMON Rx packets with > 2048 bytes (RMON_R_P_GTE2048)
0xFC03_02C4	RMON Rx octets (RMON_R_OCTETS)
0xFC03_02C8	Count of received frames not counted correctly (IEEE_R_DROP)
0xFC03_02CC	Frames received OK (IEEE_R_FRAME_OK)
0xFC03_02D0	Frames received with CRC error (IEEE_R_CRC)
0xFC03_02D4	Frames received with alignment error (IEEE_R_ALIGN)
0xFC03_02D8	Receive FIFO overflow count (IEEE_R_MACERR)
0xFC03_02DC	Flow control pause frames received (IEEE_R_FDXFC)
0xFC03_02E0	Octet count for frames received without error (IEEE_R_OCTETS_OK)

19.4.2 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in EIR, 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   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 19-2. Ethernet Interrupt Event Register (EIR)

Table 19-5. EIR Field Descriptions

Field	Description
31 HBERR	Heartbeat error. Indicates TCR[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[MAX_FL] bytes.
29 BABT	Babbling transmit error. Indicates the transmitted frame length exceeds RCR[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[GTS] bit is now complete. 2) A graceful stop initiated by the setting of the TCR[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 19.5.11, "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.
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[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.

Table 19-5. EIR Field Descriptions (continued)

Field	Description
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.

19.4.3 Interrupt Mask Register (EIMR)

The EIMR register controls 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 and EIMR registers are set, an interrupt is generated. The interrupt signal remains asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

Address: 0xFC03_0008   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 19-3. Ethernet Interrupt Mask Register (EIMR)

Table 19-6. EIMR Field Descriptions

Field	Description
31–19 See Figure 19-3 and Table 19-5	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR register. The corresponding EIMR bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR samples the signal generated by the interrupting source. The corresponding EIR bit reflects the state of the interrupt signal even if the corresponding EIMR bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
18–0	Reserved, must be cleared.

19.4.4 Receive Descriptor Active Register (RDAR)

RDAR 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[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 register is cleared at reset and when ECR[ETHER_EN] is cleared.

Address: 0xFC03_0010   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
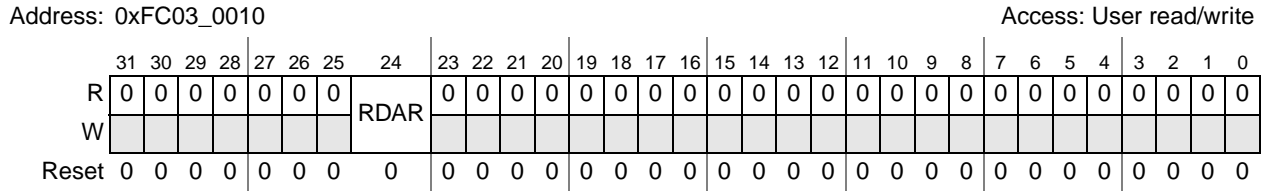
Figure 19-4. Receive Descriptor Active Register (RDAR)

Table 19-7. RDAR 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[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

19.4.5 Transmit Descriptor Active Register (TDAR)

The TDAR is a command register 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[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 register is cleared at reset, when ECR[ETHER_EN] is cleared, or when ECR[RESET] is set.

Address: 0xFC03_0014   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
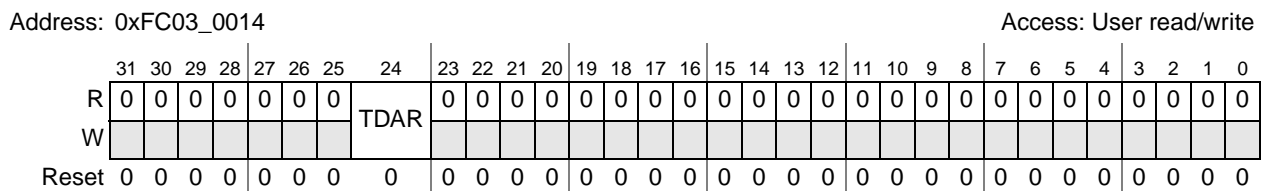
Figure 19-5. Transmit Descriptor Active Register (TDAR)

Table 19-8. TDAR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.

Table 19-8. TDAR Field Descriptions (continued)

Field	Description
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[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

19.4.6   Ethernet Control Register (ECR)

ECR is a read/write user register, though hardware may alter fields in this register as well. The ECR enables/disables the FEC.

Address: 0xFC03_0024   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
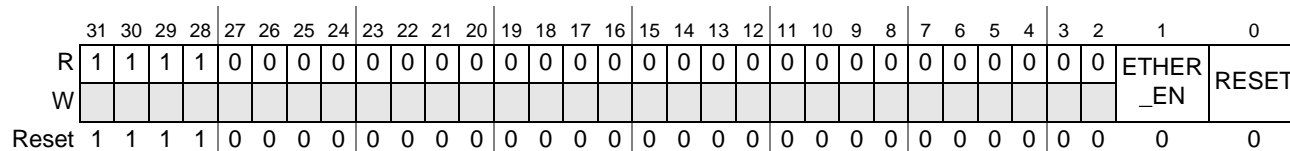
Figure 19-6. Ethernet Control Register (ECR)

Table 19-9. ECR Field Descriptions

Field	Description
31–2	Reserved, must be cleared.
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[RESET] is set by software, in which case ETHER_EN is cleared • An error condition causes the EIR[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[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.

19.4.7   MII Management Frame Register (MMFR)

The MMFR is user-accessible and does not reset to a defined value. The MMFR register is 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 causes a management frame to be sourced unless the MSCR is programmed to 0. If MSCR is cleared while MMFR is written and then MSCR is written with a non-zero value, an MII frame is generated with the data previously written to the MMFR. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.

Address: 0xFC03_0040   Access: User read/write
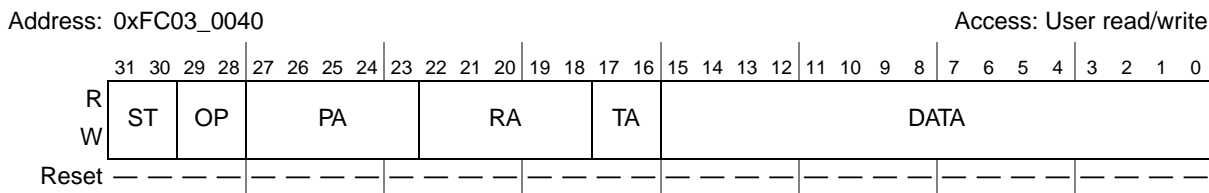
Figure 19-7. MII Management Frame Register (MMFR)

Table 19-10. MMFR 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.
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 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 register. Writing this pattern causes the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time, contents of the MMFR 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 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 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 register following a preamble generated by the control state machine. During this time, contents of the MMFR 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 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 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 register while frame generation is in progress.

19.4.8 MII Speed Control Register (MSCR)

The MSCR provides control of the MII clock (FEC_MDC pin) frequency and allows a preamble drop on the MII management frame.

Address: 0xFC03_0044   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
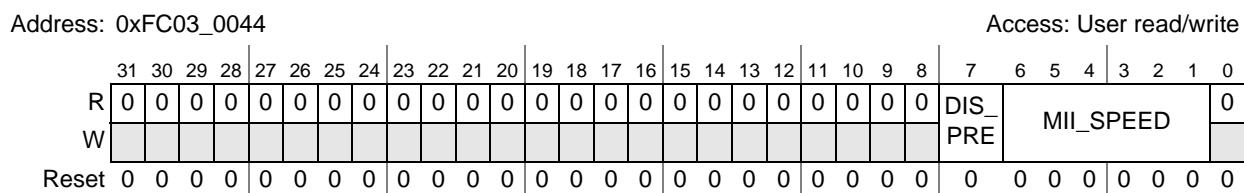
Figure 19-8. MII Speed Control Register (MSCR)

Table 19-11. 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_MDC) relative to the internal bus clock. A value of 0 in this field turns off the FEC_MDC and leaves it in low voltage state. Any non-zero value results in the FEC_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_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 register may optionally be set to 0 to turn off the FEC_MDC. The FEC_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_MDC).

If the internal bus clock is 25 MHz, programming this register to 0x0000_0005 results in an FEC_MDC as stated the equation below.

$$25 \text{ MHz} \times \frac{1}{5 \times 2} = 2.5 \text{ MHz} \qquad\qquad \textit{Eqn. 19-1}$$

A table showing optimum values for MII_SPEED as a function of internal bus clock frequency is provided below.

Table 19-12. Programming Examples for MSCR

Internal FEC Clock Frequency	MSCR[MII_SPEED]	FEC_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
80 MHz	0x10	2.50 MHz

19.4.9 MIB Control Register (MIBC)

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 19-4 for the locations of the MIB counters.

Address: 0xFC03_0064   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
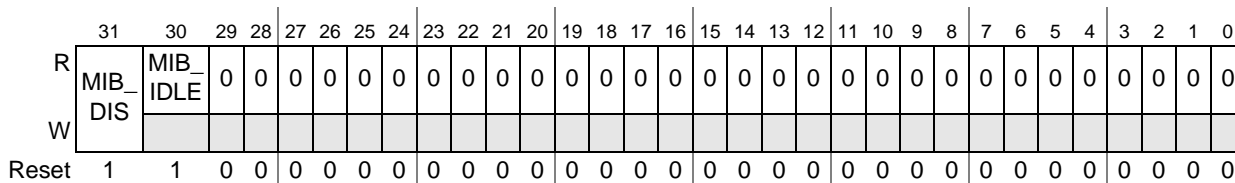
Figure 19-9. MIB Control Register (MIBC)

Table 19-13. MIBC 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.

19.4.10 Receive Control Register (RCR)

RCR controls the operational mode of the receive block and must be written only when ECR[ETHER_EN] is cleared (initialization time).

Address: 0xFC03_0084   Access: User read/write

Figure 19-10. Receive Control Register (RCR)

Table 19-14. RCR 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.
15–6	Reserved, must be cleared.
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 mode
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_TXCLK when LOOP is asserted. DRT must be set to 0 when setting LOOP.

19.4.11 Transmit Control Register (TCR)

TCR 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[ETHER_EN] is cleared.

Address: 0xFC03_00C4   Access: User read/write

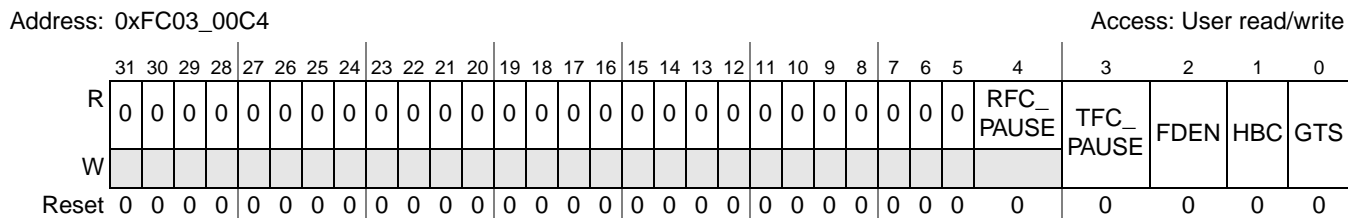|	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RFC_PAUSE	TFC_PAUSE	FDEN	HBC	GTS
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-11. Transmit Control Register (TCR)

Table 19-15. 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 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.
2 FDEN	Full duplex enable. If set, frames transmit independent of carrier sense and collision inputs. This bit should only be modified when ECR[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[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 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[ETHER_EN] following the GRA interrupt.

19.4.12 Physical Address Lower Register (PALR)

PALR 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.

Address: 0xFC03_00E4   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															PADDR1																	
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
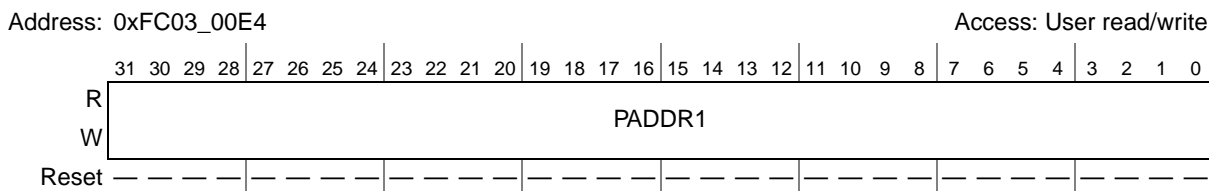
Figure 19-12. Physical Address Lower Register (PALR)

Table 19-16. PALR 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.

19.4.13 Physical Address Upper Register (PAUR)

PAUR 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 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   Access: User read/write
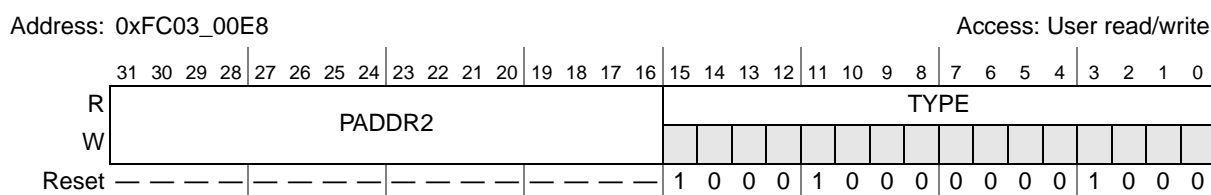
Figure 19-13. Physical Address Upper Register (PAUR)

Table 19-17. PAUR 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.

19.4.14 Opcode/Pause Duration Register (OPD)

The OPD 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   Access: User read/write
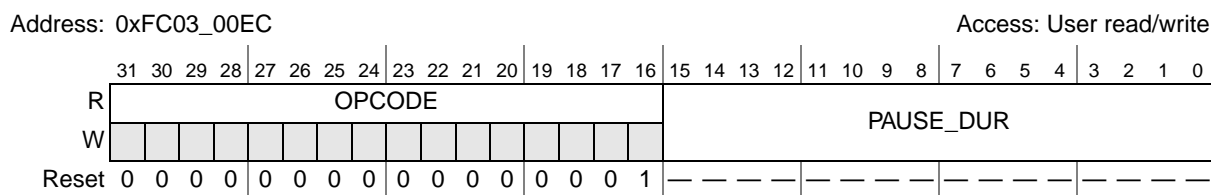
Figure 19-14. Opcode/Pause Duration Register (OPD)

Table 19-18. OPD 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.

19.4.15 Descriptor Individual Upper Address Register (IAUR)

IAUR 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　　　　　　　　　　　　　　　　　　　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					IADDR1			
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —
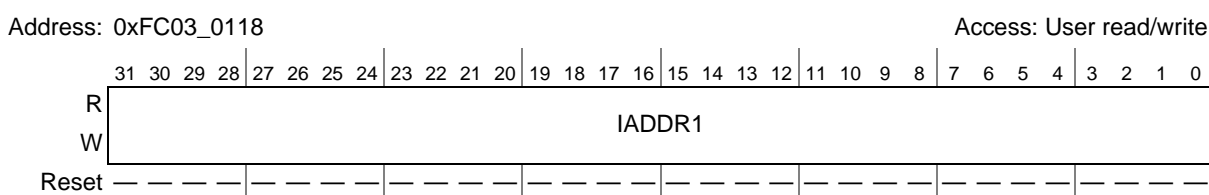
Figure 19-15. Descriptor Individual Upper Address Register (IAUR)

Table 19-19. IAUR 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.

19.4.16 Descriptor Individual Lower Address Register (IALR)

IALR 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　　　　　　　　　　　　　　　　　　　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					IADDR2			
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 19-16. Descriptor Individual Lower Address Register (IALR)

Table 19-20. IALR 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.

19.4.17 Descriptor Group Upper Address Register (GAUR)

GAUR 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   Access: User read/write

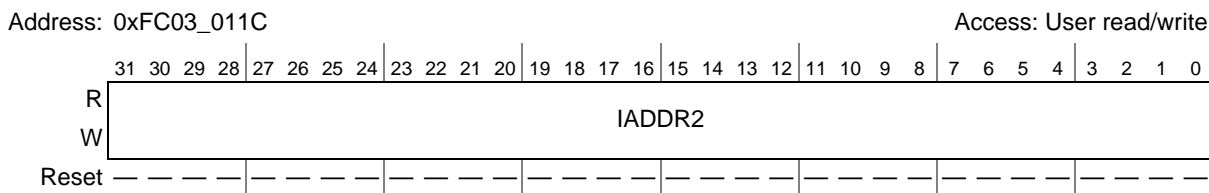|	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R					GADDR1			
W								
Reset	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —

Figure 19-17. Descriptor Group Upper Address Register (GAUR)

Table 19-21. GAUR 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.

19.4.18 Descriptor Group Lower Address Register (GALR)

GALR 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.
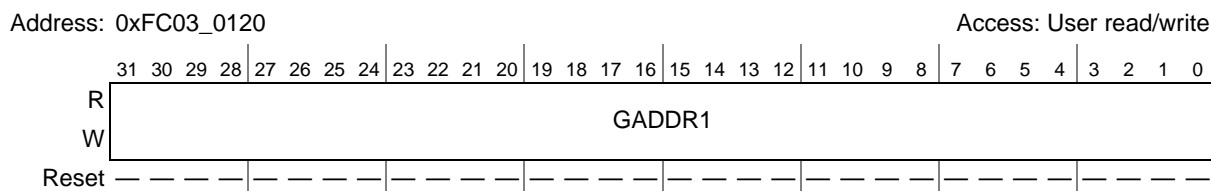
Address: 0xFC03_0124   Access: User read/write

	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	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —	— — — —
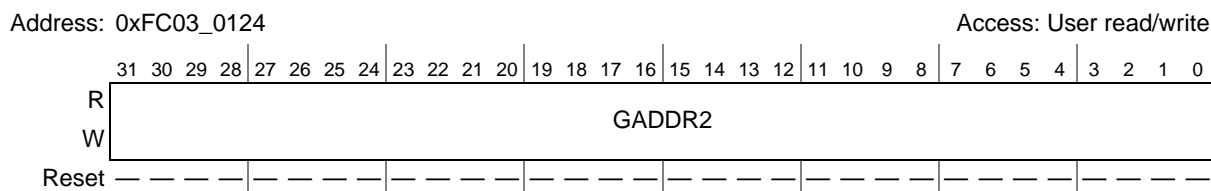
Figure 19-18. Descriptor Group Lower Address Register (GALR)

Table 19-22. GALR 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.

19.4.19 Transmit FIFO Watermark Register (TFWR)

The TFWR 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   Access: User read/write

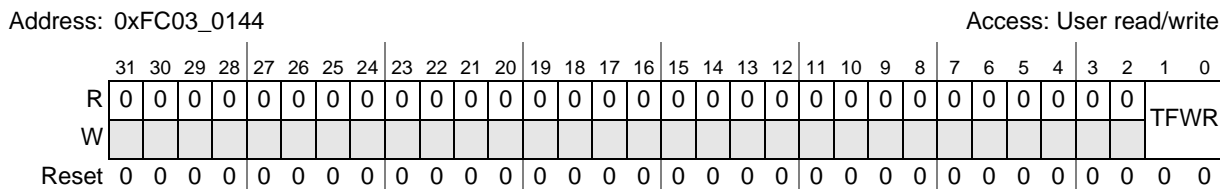|	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TFWR	
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-19. Transmit FIFO Watermark Register (TFWR)

Table 19-23. TFWR 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

19.4.20 FIFO Receive Bound Register (FRBR)

FRBR indicates the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR, to appropriately divide the available FIFO RAM between the transmit and receive data paths.
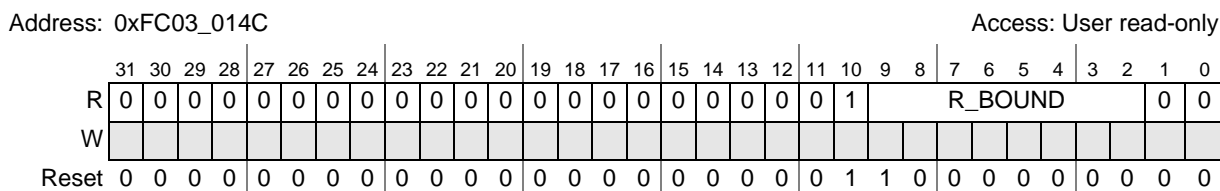
Address: 0xFC03_014C   Access: User read-only

	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 19-20. FIFO Receive Bound Register (FRBR)

Table 19-24. FRBR 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.

19.4.21 FIFO Receive Start Register (FRSR)

FRSR indicates the starting address of the receive FIFO. FRSR 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. The receive FIFO uses addresses from FRSR to FRBR inclusive.

Hardware initializes the FRSR register at reset. FRSR only needs to be written to change the default value.
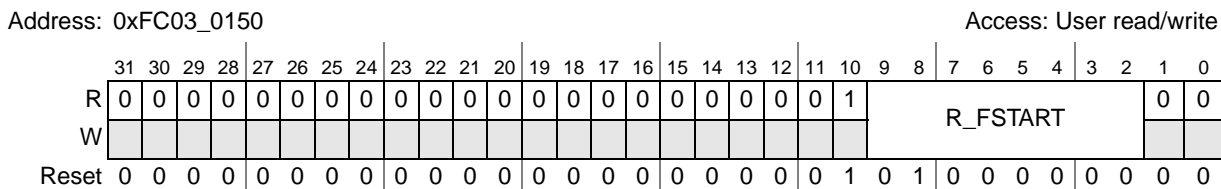
Address: 0xFC03_0150   Access: User read/write

	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 19-21. FIFO Receive Start Register (FRSR)

Table 19-25. FRSR 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.

19.4.22 Receive Descriptor Ring Start Register (ERDSR)

ERDSR 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.
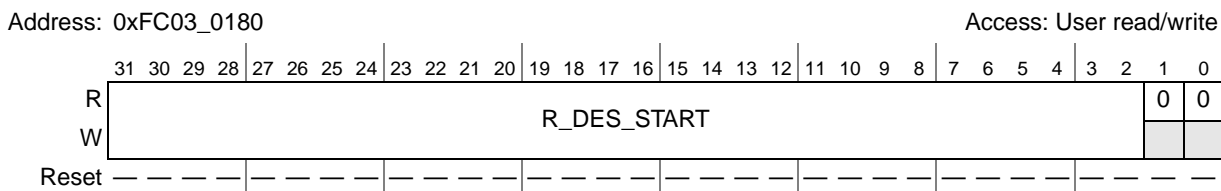
Address: 0xFC03_0180   Access: User read/write

	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 19-22. Ethernet Receive Descriptor Ring Start Register (ERDSR)

Table 19-26. ERDSR Field Descriptions

Field	Description
31–2 R_DES_START	Pointer to start of receive buffer descriptor queue.
1–0	Reserved, must be cleared.

19.4.23 Transmit Buffer Descriptor Ring Start Registers (ETSDR)

ETSDR 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.

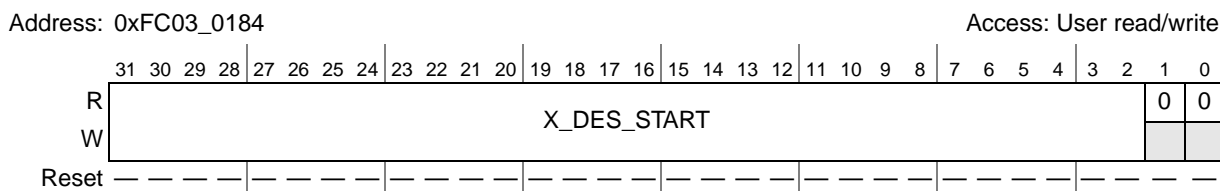This register is undefined at reset and must be initialized prior to operation.

Address: 0xFC03_0184   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 19-23. Transmit Buffer Descriptor Ring Start Register (ETDSR)

Table 19-27. ETDSR Field Descriptions

Field	Description
31–2 X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, must be cleared.

19.4.24 Receive Buffer Size Register (EMRBR)

The EMRBR 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 must be set to RCR[MAX_FL] or larger. To properly align the buffer, EMRBR 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 be greater than or equal to 256 bytes.

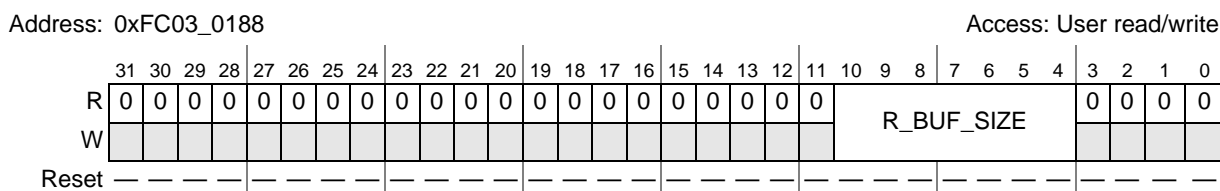The EMRBR register is undefined at reset and must be initialized by the user.

Address: 0xFC03_0188   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 19-24. Receive Buffer Size Register (EMRBR)

Table 19-28. EMRBR 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.

19.5 Functional Description

This section describes the operation of the FEC, 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 FEC.

19.5.1 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.

19.5.1.1 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[E] or TxBD[R] bit produces the buffer. Software writing to TDAR or RDAR 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[E] or TxBD[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[ETHER_EN] bit operates as a reset to the BD/DMA logic. When ECR[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[ETHER_EN] is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR 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 and ETDSR 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.

19.5.1.1.1 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[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[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. 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.

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.

19.5.1.1.2 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 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 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[E] is clear a second time, FEC stops reading receive BDs until the driver writes to RDAR.

19.5.1.2   Ethernet Receive Buffer Descriptor (RxBD)

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]															
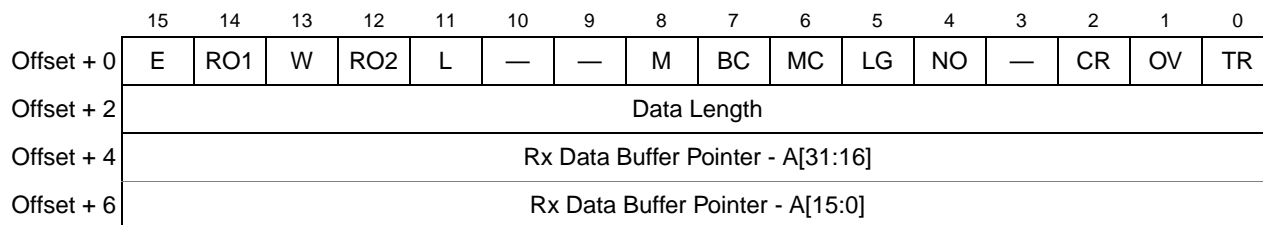
Figure 19-25. Receive Buffer Descriptor (RxBD)

Table 19-29. 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.
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.
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).

Table 19-29. Receive Buffer Descriptor Field Definitions (continued)

Word	Field	Description
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[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.

19.5.1.3 Ethernet Transmit Buffer Descriptor (TxBD)

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[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 19.4.1, "MIB Block Counters Memory Map," for more details.
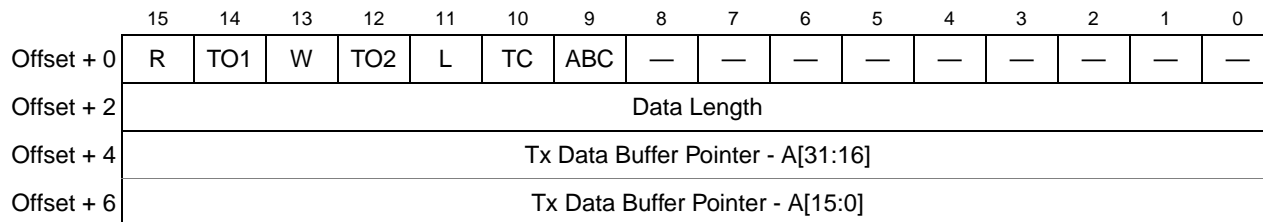
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	R	TO1	W	TO2	L	TC	ABC	—	—	—	—	—	—	—	—	—
Offset + 2	Data Length															
Offset + 4	Tx Data Buffer Pointer - A[31:16]															
Offset + 6	Tx Data Buffer Pointer - A[15:0]															

Figure 19-26. Transmit Buffer Descriptor (TxBD)

Table 19-30. 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.
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.
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 that triggers the FEC to poll the next BD in the ring.

19.5.2   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 FEC.

19.5.2.1   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[ETHER_EN] bit is cleared (which is accomplished by a hard reset or software to halt operation). By clearing ECR[ETHER_EN], configuration control registers such as the TCR and RCR are not reset, but the entire data path is reset.

Table 19-31. ECR[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	Cleared
TDAR	Cleared
Descriptor Controller block	Halt operation

19.5.3   User Initialization (Prior to Setting ECR[ETHER_EN])

You need to initialize portions the FEC prior to setting the ECR[ETHER_EN] bit. The exact values depend on the particular application. The sequence is not important.

Table 19-32 defines Ethernet MAC registers requiring initialization.

Table 19-32. User Initialization (Before ECR[ETHER_EN])

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR

Table 19-32. User Initialization (Before ECR[ETHER_EN]) (continued)

Description
PALR / PAUR (only needed for full duplex flow control)
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)
Clear MIB_RAM

Table 19-33 defines FEC FIFO/DMA registers that require initialization.

Table 19-33. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

19.5.4 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, hardware is ready for operation.

Table 19-34 shows microcontroller initialization operations.

Table 19-34. Microcontroller Initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

19.5.5   User Initialization (After Setting ECR[ETHER_EN])

After setting ECR[ETHER_EN], you can set up the buffer/frame descriptors and write to TDAR and RDAR. Refer to Section 19.5.1, "Buffer Descriptors," for more details.

19.5.6   Network Interface Options

The FEC supports an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The RCR[MII_MODE] bit select the interface mode. In MII mode (RCR[MII_MODE] set), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. Table 19-35 shows these signals.

Table 19-35. MII Mode

Signal Description	EMAC pin
Transmit Clock	FEC_TXCLK
Transmit Enable	FEC_TXEN
Transmit Data	FEC_TXD[3:0]
Transmit Error	FEC_TXER
Collision	FEC_COL
Carrier Sense	FEC_CRS
Receive Clock	FEC_RXCLK
Receive Data Valid	FEC_RXDV
Receive Data	FEC_RXD[3:0]
Receive Error	FEC_RXER
Management Data Clock	FEC_MDC
Management Data Input/Output	FEC_MDIO

The 7-wire serial mode interface (RCR[MII_MODE] cleared) is generally referred to as AMD mode. Table 19-36 shows the 7-wire mode connections to the external transceiver.

Table 19-36. 7-Wire Mode Configuration

Signal description	EMAC Pin
Transmit Clock	FEC_TXCLK
Transmit Enable	FEC_TXEN
Transmit Data	FEC_TXD[0]
Collision	FEC_COL
Receive Clock	FEC_RXCLK

Table 19-36. 7-Wire Mode Configuration (continued)

Signal description	EMAC Pin
Receive Data Valid	FEC_RXDV
Receive Data	FEC_RXD[0]

19.5.7   FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. After ECR[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), MAC transmit logic asserts FEC_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_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 19.5.15.1, "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 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[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[GTS] is cleared, the FEC resumes transmission with the next frame.

19.5.7.1   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.

19.5.8   FEC Frame Reception

The FEC receiver works 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[ETHER_EN], it immediately starts processing receive frames. When FEC_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_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 19.5.15.2, "Reception Errors," for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR 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) is set. See Section 19.5.1.2, "Ethernet Receive Buffer Descriptor (RxBD)," for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR, maskable by RFIEN bit in EIMR), indicating that a frame is received and is in memory. The Ethernet controller then waits for a new frame.

19.5.9   Ethernet Address Recognition

The FEC filters 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 19-27 illustrates the address recognition decisions made by the receive block, while Figure 19-28 illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC_REJ]) is cleared, then the frame is accepted unconditionally, as shown in Figure 19-27. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in Figure 19-28.

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 and GALR. 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 and PAUR 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 and IALR. 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 19-27.

If neither a hash match (group or individual) nor an exact match (group or individual) occur, and if promiscuous mode is enabled (RCR[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[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.
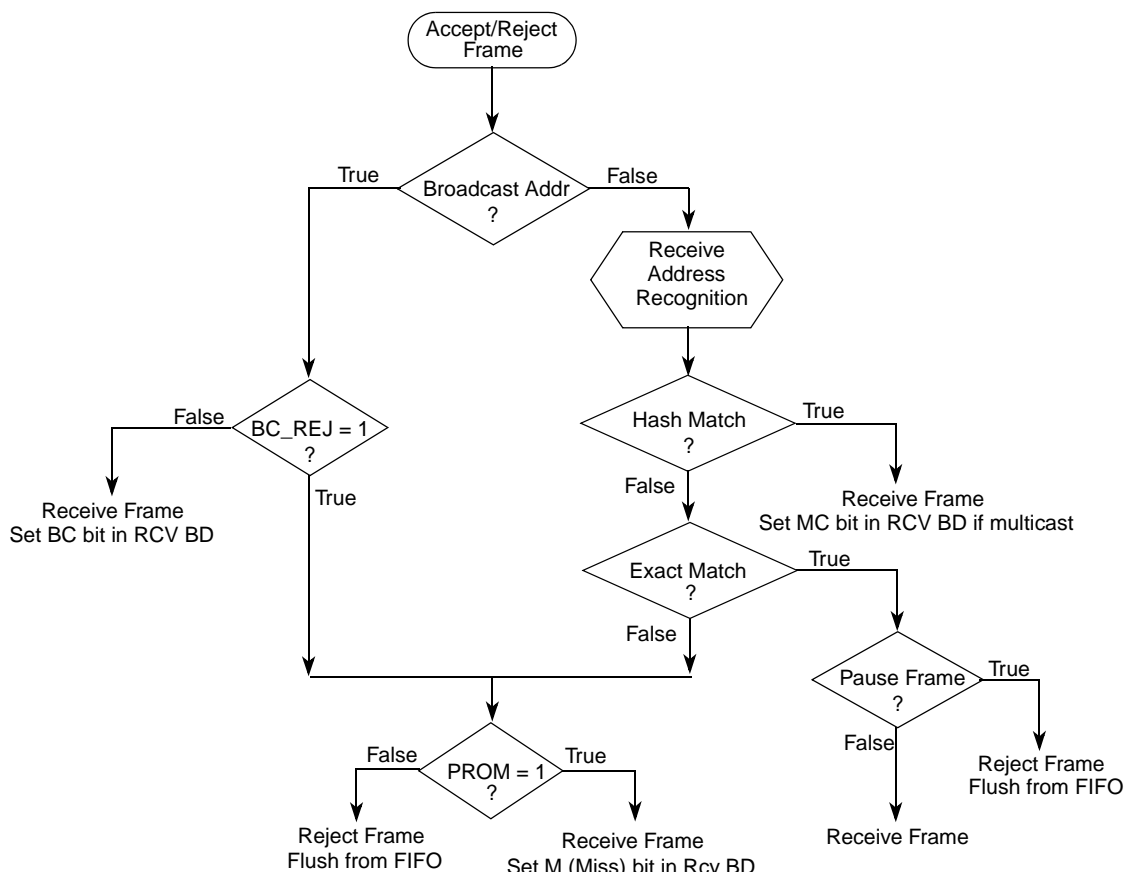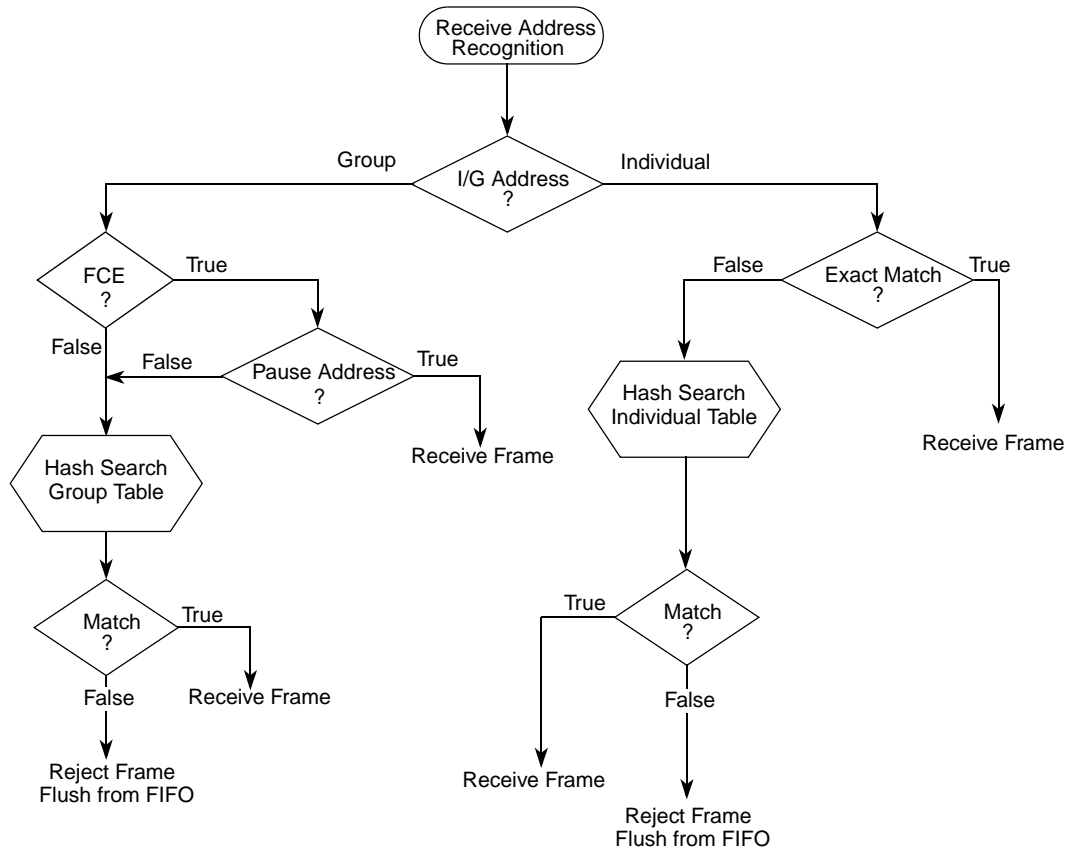

Notes:
BC_REJ - field in RCR register (BroadCast REJect)
PROM - field in RCR register (PROMiscous mode)
Pause Frame - valid PAUSE frame received

Figure 19-27. Ethernet Address Recognition—Receive Block Decisions

Notes:
FCE - field in RCR register (flow control enable)
I/G - Individual/Group bit in destination address (lsb in first byte received in MAC frame)

Figure 19-28. Ethernet Address Recognition—Microcode Decisions

19.5.10 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, GALR (group address hash match), or IAUR, IALR (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 (msb = 1) or GALR (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:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1 \qquad \textbf{\textit{Eqn. 19-2}}$$

Table 19-37 contains example destination addresses and corresponding hash values.

Table 19-37. 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 19-37. 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 19-37. 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

19.5.11 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[FDEN] set) with flow control (RCR[FCE] set). The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in Table 19-38. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 19-38. 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[GTS] is set by the FEC internally. When transmission has paused, the EIR[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[PAUSE_DUR] slot times have expired. On OPD[PAUSE_DUR] expiration, TCR[GTS] is cleared allowing MAC data frame transmission to resume. The receive flow control pause status bit (TCR[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[TFC_PAUSE]). After TCR[TFC_PAUSE] is set, the transmitter sets TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts and the pause frame is transmitted. TCR[TFC_PAUSE,GTS] are then cleared internally.

You must specify the desired pause duration in the OPD register.

When the transmitter pauses due to receiver/microcontroller pause frame detection, TCR[TFC_PAUSE] may remain set and cause the transmission of a single pause frame. In this case, the EIR[GRA] interrupt is not asserted.

19.5.12 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.

19.5.13 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.

19.5.14 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[LOOP, DRT] and TCR[FDEN] bits.

Set FDEN for internal and external loopback.

For internal loopback, set RCR[LOOP] and clear RCR[DRT]. FEC_TXEN and FEC_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[LOOP] and RCR[DRT], and configure the external transceiver for loopback.

19.5.15 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 register.

19.5.15.1 Transmission Errors

19.5.15.1.1 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[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 register.

19.5.15.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and EIR[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 register.

19.5.15.1.3 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[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 register.

19.5.15.1.4 Heartbeat

Some transceivers have a self-test feature 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[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[HB], and generates the HBERR interrupt if it is enabled.

19.5.15.2 Reception Errors

19.5.15.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, FEC sets RxBD[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.

19.5.15.2.2 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. If there is no CRC error, no error is reported.

19.5.15.2.3 CRC Error

When a CRC error occurs with no dribble bits, FEC closes the buffer and sets RxBD[CR]. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

19.5.15.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes the BABR interrupt is generated, and RxBD[LG] is set. The frame is not truncated unless the frame length exceeds 2047 bytes.

19.5.15.2.5 Truncation

When the receive frame length exceeds 2047 bytes, frame is truncated and RxBD[TR] is set.

Chapter 20
Universal Serial Bus Interface – Host Module

This chapter describes the universal serial bus (USB) host module, which implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications. Readers of this chapter are assumed to be fluent in the operation and requirements of a USB network.

The following documents are available from the USB Implementers Forum web page at http://www.usb.org/developers/docs:

- *Universal Serial Bus Specification, Revision 2.0*

The following documents are available from the Intel USB Specifications web page at http://www.intel.com/technology/usb/spec.htm:

- *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

20.1   Introduction

The processor implements two USB modules: a host module and an On-The-Go (OTG) module. The host module is used with a full-speed/low-speed on-chip transceiver or . Whereas, the OTG module can be used with the on-chip transceiver or with an external ULPI transceiver. For more details on the USB OTG module, refer to Chapter 21, "Universal Serial Bus Interface – On-The-Go Module."

USB host modules must supply 500 mA with a 5 V supply on its downstream port (referred to as VBUS). If the connected device attempts to draw more than the allocated amount of current, the USB host must disable the port and remove power. This processor provides pins for control and status to an external IC capable of managing the VBUS downstream supply.

Register and data structure interfaces conform to the EHCI specification from Intel Corporation, with enhancements to support the embedded environment. The USB controller contains its own DMA (direct memory access) engines that reduce interrupt load on the application processor, and thereby reduce total system bus bandwidth dedicated to servicing the USB interface requirements. The USB controller includes logic to support USB's low-power suspend features, and for suspended devices to request remote wakeup from the host.

This USB host controller hides all direct interaction with the protocol, but some knowledge of the USB is required to properly configure the device for operation on the local bus and on the USB. This document covers programming requirements, and additional information may be found in the USB specification.

20.1.1 Block Diagram
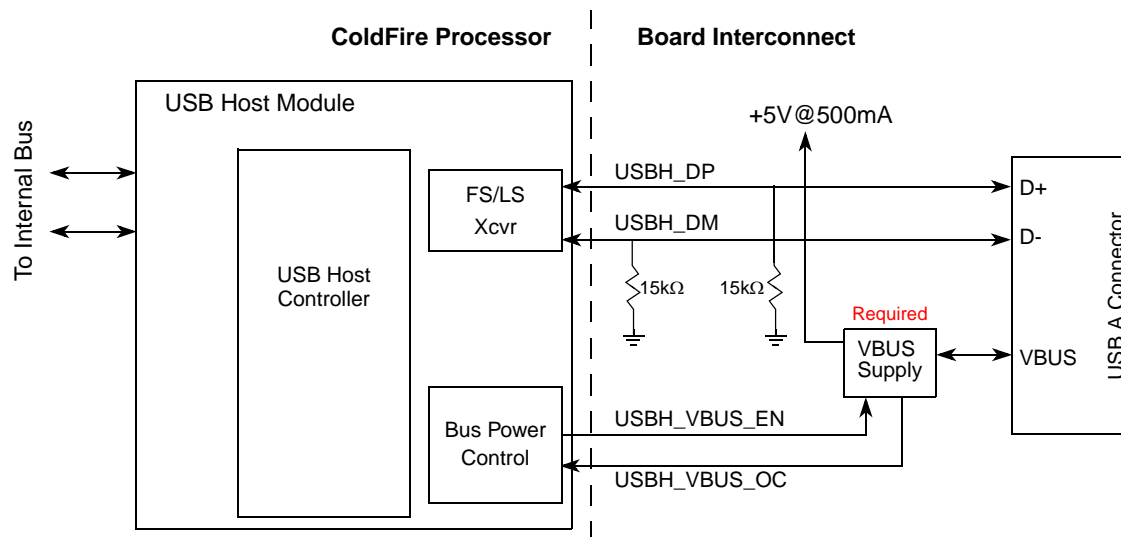
The USB host module is shown below in Figure 20-1.

Figure 20-1. USB Host Interface Block Diagram

20.1.2 Overview

The USB host module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures for both modules are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* from Intel Corporation. The USB controller supports directly connected full- and low-speed peripherals without the need for UHCI or OHCI companion controllers.

The USB host module interfaces to the processor's ColdFire core. The USB controller is programmable to support full-speed (12 Mbps) and low-speed (1.5 Mbps) applications using the on-chip transceiver. The processor's on-chip PLL provides all necessary clocks to the USB controller. For special applications, pin access (via USBCLKIN) is provided for an external USB reference clock (60MHz).

The USB host controller provides control and status signals to interface with external USB host power devices. Use these control and status signals on the chip interface to communicate with external USB host power solutions. USB VBUS is not provided on-chip.

The on-chip FS/LS transceiver does not include USB DP and DM bias resistors. 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 this device as they are available via a standard product from various manufacturers.

20.1.3 Features

The USB host module includes the following features:

- Complies with USB specification revision 2.0
- Supports operation as a standalone USB host controller

MCF5329 Reference Manual, Rev 3

— Supports enhanced host controller interface (EHCI)

— Allows direct connection of full-speed (FS) or low-speed (LS) devices without an OHCI/UHCI companion controller

— Supported by Linux and other commercially available operating systems

- Includes on-chip full-speed (12 Mbps), and low-speed (1.5 Mbps) transceiver

- Allows vendor to define any USB device or class for targeted peripheral list, including USB hubs

- Supports VBUS power enable and VBUS over-current detect to control bus power

- Registers provided for indicating VBUS state to controller

- 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

— Integrated with the processor's doze and stop modes for ultra-low power operation

20.1.4 Modes of Operation

The USB host controller provides the functionality of one USB 2.0 host. The module is hardwired to an on-chip full-/low-speed transceiver. Speed selection is auto-detected at connect time via sensing of the DP or DM pullup resistor on the connected device using procedures of enumeration in the USB network. The USB host module provides the following modes of operation for the user:

- USB disabled. In this mode, the USB host 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 20.1.4.1, "Low-Power Modes," for details.

20.1.4.1 Low-Power Modes

The USB host module is integrated with the Version 3 ColdFire core's low-power modes (stop, doze and wait). The modes are implemented as follows:

- Stop — The processor stops the clock to the USB host module. In this state, the module ignores traffic on the USB network 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 host module are running.

- Doze — The processor stops the system clocks to the USB host module. However, the 60 Mhz transceiver clock remains active. Detection of resume signaling initiates a restart of the module clocks.

20.2 External Signal Description

Table 20-1 describes the external signals of the USB host module.

Table 20-1. USB Host External Signals

Signal	I/O	Description	
USB_CLKIN	I	Optional 60 MHz clock source.	
USBH_DM	I/O	D- output of the dual-speed transceiver for the USB Host module.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous
USBH_DP	I/O	D+ output of the dual-speed transceiver for the USB Host module.	
		State Meaning	Asserted—Data 1 Negated—Data 0
		Timing	Asynchronous
USBH_VBUS_EN	O	Enables off-chip charge pump controller of VBUS for the USB host module.	
		State Meaning	Asserted—Off-chip charge pump controller enabled Negated—Off-chip charge pump controller disabled
		Timing	Asynchronous
USBH_VBUS_OC	I	Communicates short on USB lines occurred for USB host module.	
		State Meaning	Asserted—Short detected Negated—No short detected
		Timing	Synchronous to USB_CLKIN.

20.2.1 USB Host Control and Status Signals

To minimize the pin-count on the device, a few USB host control and status signals are implemented on-chip as bit fields in a register within the chip configuration module (CCM).

The host controller status register (UHCSR) is implemented as follows:

- Writes to the UHCSR register from the firmware set the corresponding bits on the USB interface.
- When the USB host module outputs change, the corresponding bits on the UHCSR register are updated, and a maskable interrupt is generated.

The UHCSR register is documented in the CCM chapter, see Section 9.3.6, "USB Host Controller Status Register (UHCSR)."

Table 20-2. Internal Control and Status Bits for USB Host Module

Signal	Mnemonic	Direction	Access	Interrupt Trigger?
Port Indicator LED Control	PORTIND[1:0]	Selects LED color for product package.	R	N
Wake-up Event	WKUP	Reflects when a wake-up event has occurred on the USB bus.	R/W	Y

Table 20-2. Internal Control and Status Bits for USB Host Module (continued)

Signal	Mnemonic	Direction	Access	Interrupt Trigger?
Interrupt Mask	UHMIE	Interript enable. When set, changes on WKUP cause an interrupt to be asserted. When cleared, the interrupt is masked.	R/W	N/A
On-chip Transceiver Pull-down Enable	XPDE	Enables 50 kΩ pull-downs on the host controller's DM and DP pins	R/W	N

20.3 Memory Map/Register Definitions

This section provides the memory map of the USB host module. Descriptions of these registers can be found in Chapter 21, "Universal Serial Bus Interface – On-The-Go Module." See the Section/Page heading in the below table for direct links to the corresponding register description. Addresses and reset values shown here are correct for the USB host module.

Table 20-3. USB Host Controller Memory Map

Address	Register	EHCI[1]	Width (bits)	Access	Reset	Section/Page
Module Identification Registers						
0xFC0B_4000	Identification Register (ID)	N	32	R	0x0041_FA05	21.3.1.1/21-9
0xFC0B_4004	General Hardware Parameters (HWGENERAL)	N	32	R	0x0000_02C5	21.3.1.2/21-10
0xFC0B_4008	Host Hardware Parameters (HWHOST)	N	32	R	0x1002_0001	21.3.1.3/21-11
0xFC0B_4010	TX Buffer Hardware Parameters (HWTXBUF)	N	32	R	0x8004_0404	21.3.1.5/21-12
0xFC0B_4014	RX Buffer Hardware Parameters (HWRXBUF)	N	32	R	0x0000_0404	21.3.1.6/21-12
Capability Registers						
0xFC0B_4100	Host Interface Version Number (HCIVERSION)	Y	16	R	0x0100	21.3.2.1/21-13
0xFC0B_4103	Capability Register Length (CAPLENGTH)	Y	8	R	0x40	21.3.2.4/21-15
0xFC0B_4104	Host Structural Parameters (HCSPARAMS)	Y	32	R	0x0001_0011	21.3.2.3/21-14
0xFC0B_4108	Host Capability Parameters (HCCPARAMS)	Y	32	R	0x0000_0006	21.3.2.4/21-15
Operational Registers						
0xFC0B_4140	USB Command (USBCMD)	Y	32	R/W	0x0008_0B00	21.3.3.1/21-17
0xFC0B_4144	USB Status (USBSTS)	Y	32	R/W	0x0000_1000	21.3.3.2/21-19
0xFC0B_4148	USB Interrupt Enable (USBINTR)	Y	32	R/W	0x0000_0000	21.3.3.3/21-21
0xFC0B_414C	USB Frame Index (FRINDEX)	Y	32	R/W	0x0000_0000	21.3.3.4/21-23
0xFC0B_4154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	32	R/W	0x0000_0000	21.3.3.5/21-24
0xFC0B_4158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	32	R/W	0x0000_0000	21.3.3.7/21-25

Table 20-3. USB Host Controller Memory Map (continued)

Address	Register	EHCI[1]	Width (bits)	Access	Reset	Section/Page
0xFC0B_415C	Host TT Asynchronous Buffer Control (TTCTRL)	N	32	R/W	0x0000_0000	21.3.3.9/21-26
0xFC0B_4160	Master Interface Data Burst Size (BURSTSIZE)	N	32	R/W	0x0000_0404	21.3.3.10/21-27
0xFC0B_4164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	32	R/W	0x0002_0000	21.3.3.11/21-27
0xFC0B_4180	Configure Flag Register (CONFIGFLAG)	Y	32	R	0x0000_0001	21.3.3.13/21-31
0xFC0B_4184	Port Status/Control (PORTSC1)	Y	32	R/W	0xEC00_0000	21.3.3.14/21-31
0xFC0B_41A8	USB Mode Register (MODE)	N	32	R/W	0x0000_0003	21.3.3.16/21-38

[1] Indicates if the register is present in the EHCI specification.

20.4 Functional Description

The USB host module's functional description is very similar to the USB OTG module in host mode. See Chapter 21, "Universal Serial Bus Interface – On-The-Go Module," and the *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0* for more information.

Chapter 21
Universal Serial Bus Interface – On-The-Go Module

21.1 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*

21.1.1 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^2C 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 does not include USB DP and DM bias resistors. 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.

21.1.2   Block Diagram

Figure 21-1 shows the USB On-The-Go interface using the on-chip full-speed/low-speed transceiver.
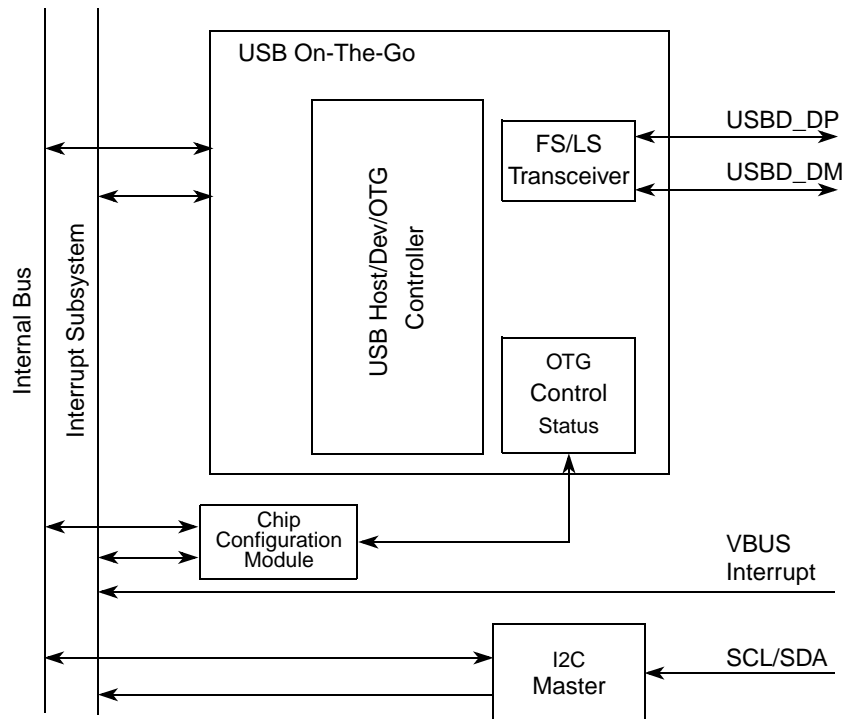
Figure 21-1. USB On-The-Go with on-chip FS/LS Transceiver Interface Block Diagram

Figure 21-2 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 21-1. 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.
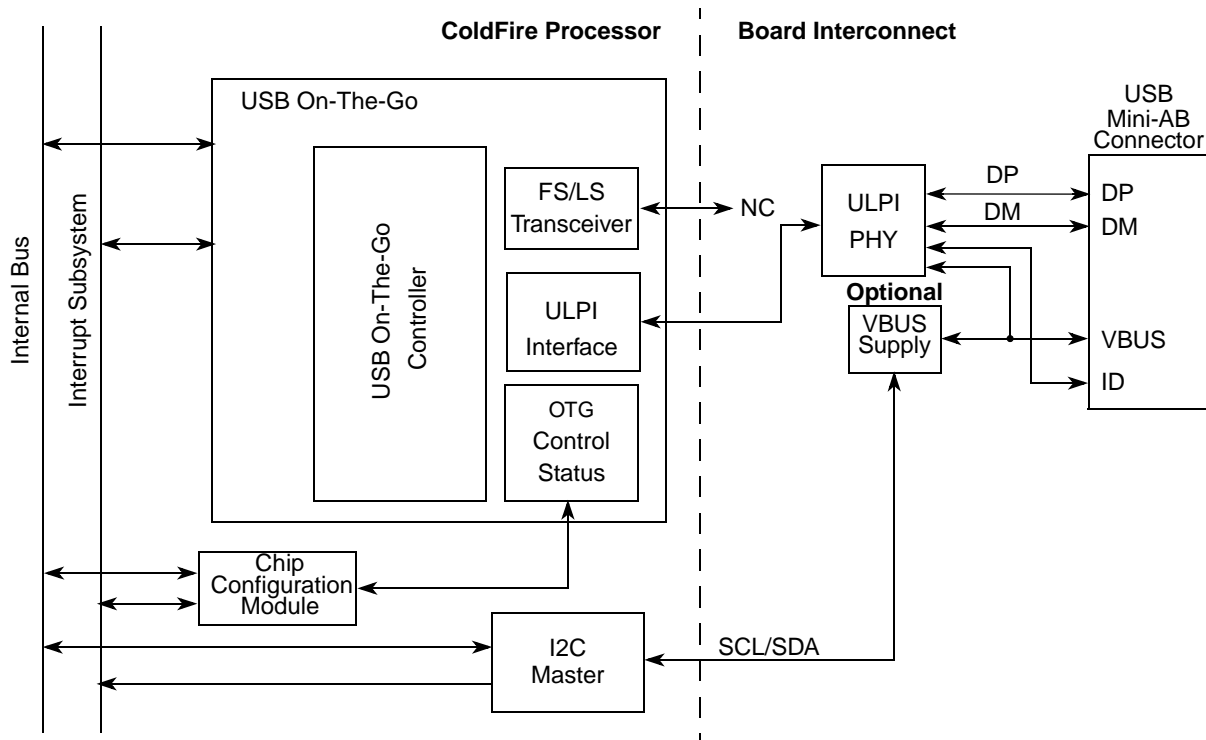
Figure 21-2. USB On-The-Go module and ULPI transceiver/PHY

21.1.3   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 21-1. 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.

21.1.4 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.

The USB OTG module is configured to use the on-chip FS/LS transceiver by default. The OTG module also includes an interface to an optional, external high-speed ULPI PHY. For high-speed operation, the external ULPI transceiver must be used. Selection of the on-chip transceiver versus the ULPI PHY is accomplished via programming of PORTSC1 register.

NOTE

Due to package limitations, the ULPI interface is not available in all pin configurations.

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 21.1.4.1, "Low-Power Modes," for details.

21.1.4.1 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.

21.2 External Signal Description

Table 21-2 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 13, "General Purpose I/O Module," for more details.

Table 21-2. 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.	
USBOTG_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
USBOTG_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
USBOTG_PU_EN	O	Enables an external pull-up on the USBOTG_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			

Table 21-2. USB OTG Signal Descriptions (continued)

Signal	I/O	Description
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.
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

21.2.1 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^2C 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 9.3.7, "USB On-the-Go Controller Status Register (UOCSR)."

Table 21-3. Internal Control and Status Bits for USB OTG Module

Signal	Mnemonic	Direction	Comments	Interrupt Trigger?
Port Indicator LED Control	PORTIND[1:0]	Selects LED color for product package.	R	N
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 Drive	DRV_VBUS	Enables bus power on VBUS to connected device.	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

21.3 Memory Map/Register Definition

This section provides the memory map and detailed descriptions of all USB-interface registers. See Table 21-4 for the memory map of the USB OTG interface. Registers in USB host module are similar, starting with 0xFC0B_4*xxx*. See Chapter 20, "Universal Serial Bus Interface – Host Module," for the USB host memory map.

Table 21-4. 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	0x0041_FA05	21.3.1.1/21-9
0xFC0B_0004	General Hardware Parameters (HWGENERAL)	N	H/D	32	R	0x0000_07C5	21.3.1.2/21-10
0xFC0B_0008	Host Hardware Parameters (HWHOST)	N	H/D	32	R	0x1002_0001	21.3.1.3/21-11
0xFC0B_000C	Device Hardware Parameters (HWDEVICE)	N	D	32	R	0x0000_0009	21.3.1.4/21-11
0xFC0B_0010	TX Buffer Hardware Parameters (HWTXBUF)	N	H/D	32	R	0x8004_0604	21.3.1.5/21-12
0xFC0B_0014	RX Buffer Hardware Parameters (HWRXBUF)	N	H/D	32	R	0x0000_0404	21.3.1.6/21-12
	Capability Registers						
0xFC0B_0100	Host Interface Version Number (HCIVERSION)	Y	H	16	R	0x0100	21.3.2.1/21-13
0xFC0B_0103	Capability Register Length (CAPLENGTH)	Y	H/D	8	R	0x40	21.3.2.2/21-13
0xFC0B_0104	Host Structural Parameters (HCSPARAMS)	Y	H	32	R	0x0001_0011	21.3.2.3/21-14
0xFC0B_0108	Host Capability Parameters (HCCPARAMS)	Y	H	32	R	0x0000_0006	21.3.2.4/21-15
0xFC0B_0122	Device Interface Version Number (DCIVERSION)	N	D	16	R	0x0001	21.3.2.5/21-15
0xFC0B_0124	Device Capability Parameters (DCCPARAMS)	N	D	32	R	0x0000_0184	21.3.2.6/21-16
	Operational Registers						
0xFC0B_0140	USB Command (USBCMD)	Y	H/D	32	R/W	0x0008_0000	21.3.3.1/21-17
0xFC0B_0144	USB Status (USBSTS)	Y	H/D	32	R/W	0x0000_0080	21.3.3.2/21-19
0xFC0B_0148	USB Interrupt Enable (USBINTR)	Y	H/D	32	R/W	0x0000_0000	21.3.3.3/21-21
0xFC0B_014C	USB Frame Index (FRINDEX)	Y	H/D	32	R/W	0x0000_0000	21.3.3.4/21-23
0xFC0B_0154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	H	32	R/W	0x0000_0000	21.3.3.5/21-24
0xFC0B_0154	Device Address (DEVICEADDR)	N	D	32	R/W	0x0000_0000	21.3.3.6/21-25
0xFC0B_0158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	H	32	R/W	0x0000_0000	21.3.3.7/21-25
0xFC0B_0158	Address at Endpoint List (EPLISTADDR)	N	D	32	R/W	0x0000_0000	21.3.3.8/21-26
0xFC0B_015C	Host TT Asynchronous Buffer Control (TTCTRL)	N	H	32	R/W	0x0000_0000	21.3.3.9/21-26
0xFC0B_0160	Master Interface Data Burst Size (BURSTSIZE)	N	H/D	32	R/W	0x0000_0404	21.3.3.10/21-27
0xFC0B_0164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	H	32	R/W	0x0000_0000	21.3.3.11/21-27

Table 21-4. 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	21.3.3.12/21-29
0xFC0B_0180	Configure Flag Register (CONFIGFLAG)	Y	H/D	32	R	0x0000_0001	21.3.3.13/21-31
0xFC0B_0184	Port Status/Control (PORTSC1)	Y	H/D	32	R/W	0xEC00_0004	21.3.3.14/21-31
0xFC0B_01A4	On-The-Go Status and Control (OTGSC)	N	H/D	32	R/W	0x0000_1020	21.3.3.15/21-35
0xFC0B_01A8	USB Mode Register (MODE)	N	H/D	32	R/W	0x0000_0000	21.3.3.16/21-38
0xFC0B_01AC	Endpoint Setup Status Register (EPSETUPSR)	N	D	32	R/W	0x0000_0000	21.3.3.17/21-39
0xFC0B_01B0	Endpoint Initialization (EPPRIME)	N	D	32	R/W	0x0000_0000	21.3.3.18/21-39
0xFC0B_01B4	Endpoint De-initialize (EPFLUSH)	N	D	32	R/W	0x0000_0000	21.3.3.19/21-40
0xFC0B_01B8	Endpoint Status Register (EPSR)	N	D	32	R	0x0000_0000	21.3.3.20/21-40
0xFC0B_01BC	Endpoint Complete (EPCOMPLETE)	N	D	32	R/W	0x0000_0000	21.3.3.21/21-41
0xFC0B_01C0	Endpoint Control Register 0 (EPCR0)	N	D	32	R/W	0x0080_0080	21.3.3.22/21-42
0xFC0B_01C4	Endpoint Control Register 1 (EPCR1)	N	D	32	R/W	0x0000_0000	21.3.3.23/21-43
0xFC0B_01C8	Endpoint Control Register 2 (EPCR2)	N	D	32	R/W	0x0000_0000	21.3.3.23/21-43
0xFC0B_01CC	Endpoint Control Register 3 (EPCR3)	N	D	32	R/W	0x0000_0000	21.3.3.23/21-43

[1] Indicates if the register is present in the EHCI specification.

[2] Indicates if the register is available in host and/or device modes.

21.3.1 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.

21.3.1.1 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	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1

Figure 21-3. Identification Register (ID)

Table 21-5. 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.

21.3.1.2 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 21-4. General Hardware Parameters Register (HWGENERAL)

Table 21-6. 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; for the USB host module, always 01.
0	Reserved, always set.

21.3.1.3 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
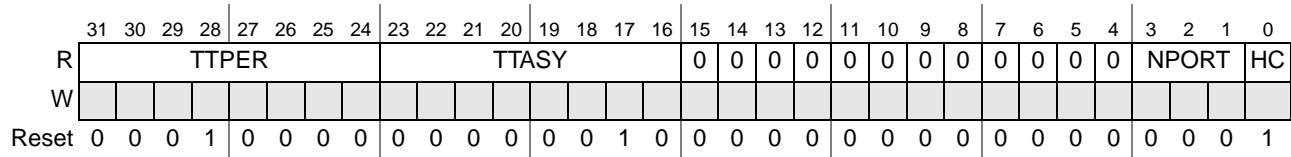
Figure 21-5. Host Hardware Parameters Register (HWHOST)

Table 21-7. 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; Always 1 for the USB host module.
0 HC	Indicates module is host capable. Always set.

21.3.1.4 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 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
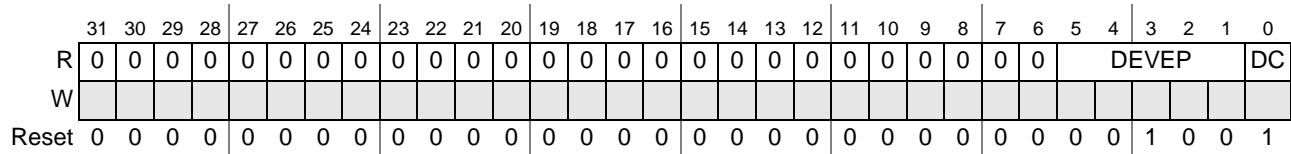
Figure 21-6. Device Hardware Parameters Register (HWDEVICE)

Table 21-8. 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.

21.3.1.5 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 21-7. Transmit Buffer Hardware Parameters Register (HWTXBUF)

Table 21-9. 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; Always clear on USB host. 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.

21.3.1.6 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 21-8. Receive Buffer Hardware Parameters Register (HWRXBUF)

Table 21-10. 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.

21.3.2 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.

21.3.2.1 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 21-9 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
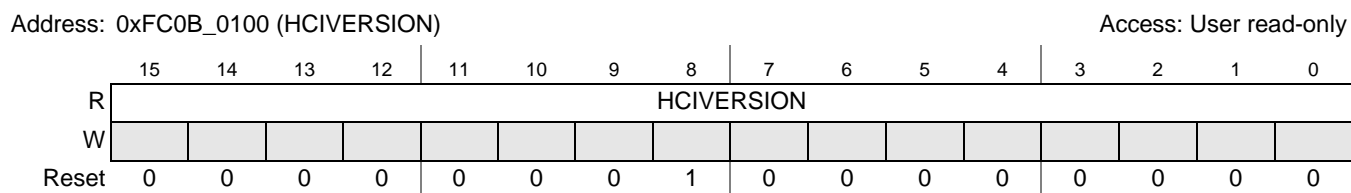
Figure 21-9. Host Controller Interface Version Register (HCIVERSION)

Table 21-11. HCIVERSION Field Descriptions

Field	Description
15–0 HCIVERSION	EHCI revision number. Value is 0x0100 indicating version 1.0.

21.3.2.2 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
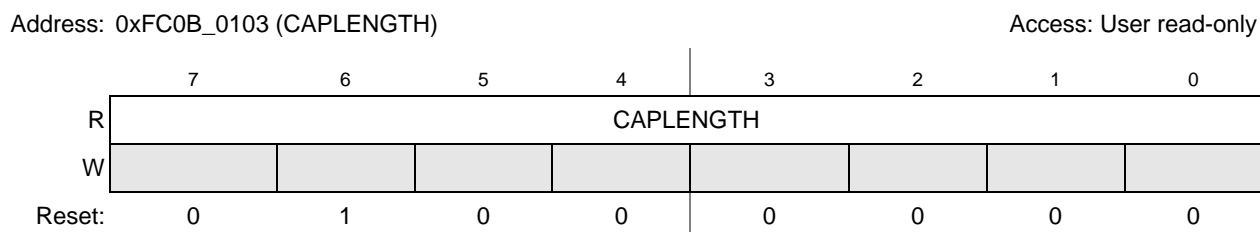
Figure 21-10. Capability Registers Length Register (CAPLENGTH)

Table 21-12. CAPLENGTH Field Descriptions

Field	Description
7–0 CAPLENGTH	Capability registers length. Always 0x40.

21.3.2.3 Host Controller Structural Parameters Register (HCSPARAMS)

This register contains structural parameters such as the number of downstream ports. Figure 21-11 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	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
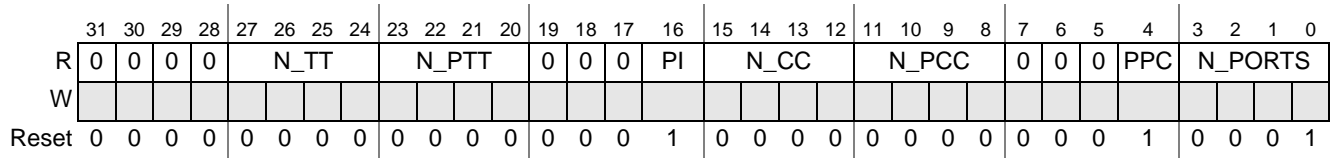
Figure 21-11. Host Controller Structural Parameters Register (HCSPARAMS)

Table 21-13. 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 21.5.5.1, "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 set. 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 21-3 for more information.
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 host and OTG modules, this is always 0x1.

21.3.2.4 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
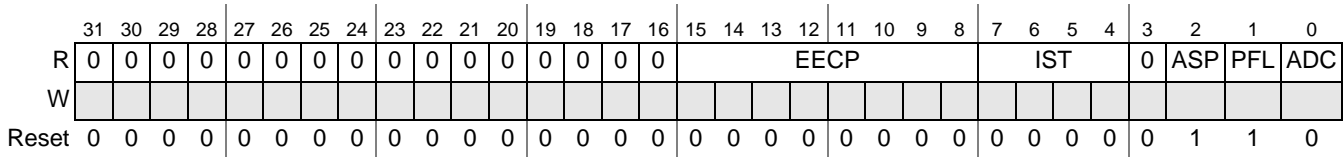
Figure 21-12. Host Controller Capability Parameters Register (HCCPARAMS)

Table 21-14. 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.
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

21.3.2.5 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_0120 (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
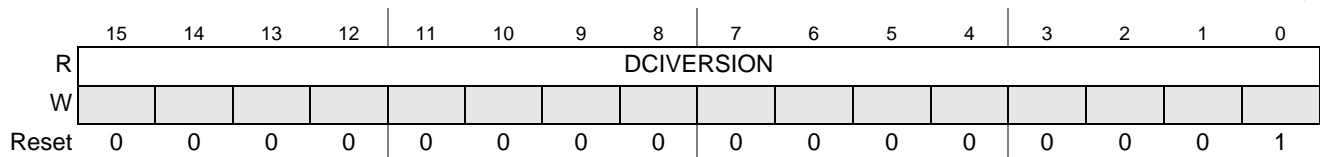
Figure 21-13. Device Controller Interface Version Register (DCIVERSION)

Table 21-15. DCIVERSION Field Descriptions

Field	Description
15–0 DCIVERSION	Device interface revision number.

21.3.2.6 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 21-14. Device Control Capability Parameters (DCCPARAMS)

Table 21-16. 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.
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.

21.3.3 Operational Registers

Comprised of dynamic control or status registers and are defined below.

21.3.3.1 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 21-15. USB Command Register (USBCMD)

Table 21-17. 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.
14 ATDTW	Add dTD TripWire. This is a non-EHCI bit, that is present on the USB OTG module only. 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 21.5.3.6.3, "Executing a Transfer Descriptor."
13 SUTW	Setup TripWire. A non-EHCI bit present on the USB OTG module only. 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 21.5.3.4.4, "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.

Table 21-17. USBCMD Field Descriptions (continued)

Field	Description
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 controling 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 21-17. 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 (USB Host and USB OTG):
	    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 (USB OTG-only):
	    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 (USB Host and USB OTG):
	    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 (USB OTG-only):
	    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.

21.3.3.2 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	0	0	0	0	0	0	0	0	0	NAKI
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	AS	PS	RCL	HCH	0	ULPII	0	SLI	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 21-16. USB Status Register (USBSTS)

Table 21-18. USBSTS Field Descriptions

Field	Description
31–17	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.
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. Present only in the OTG module.
9	Reserved, must be cleared.
8 SLI	Device-controller suspend. Non-EHCI bit present on the USB OTG module only. 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 (USB host and USB OTG):
	  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 (USB OTG-only):
	  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.

Table 21-18. USBSTS Field Descriptions (continued)

Field	Description
6 URI	USB reset received. A non-EHCI bit present on the USB OTG module only. 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.

2
PCI
	Port change detect. This bit is not EHCI compatible.
	Host mode (USB host and USB OTG):
	    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 (USB OTG only):
	    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 SLI 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 21-56 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.

21.3.3.3 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	0	0	0	0	0	0	0	0	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								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 21-17. USB Interrupt Enable Register (USBINTR)

Table 21-19. USBINTR Field Descriptions

Field	Description
31–17	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 enablePresent on the OTG module only.. 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 present on the OTG module only. When this bit is set and the USBSTS[SLI] bit transitions, USB OTG controller issues an interrupt. Software writing a 1 to the USBSTS[SLI] 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 present on the USB OTG module only. 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
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

Table 21-19. USBINTR Field Descriptions (continued)

Field	Description
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

21.3.3.4 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 (USB OTG-only), 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
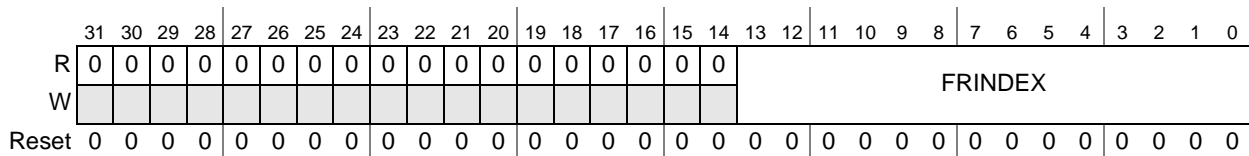
Figure 21-18. Frame Index Register (FRINDEX)

Table 21-20. 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 for the USB OTG module, 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 21-21 illustrates values of N based on the value of the USBCMD[FS] field when used in host mode.

Table 21-21. 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

21.3.3.5 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.

On the USB OTG module, 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 21.3.3.6, "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
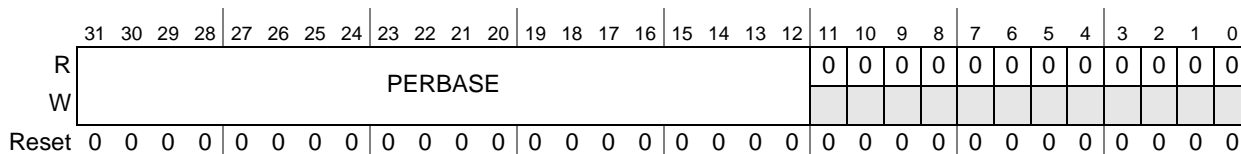
Figure 21-19. Periodic Frame List Base Address Register (PERIODICLISTBASE)

Table 21-22. 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.

21.3.3.6 Device Address Register (DEVICEADDR)

This register is not defined in the EHCI specification. For the USB OTG module in 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.

On the USB OTG module, 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 21.3.3.5, "Periodic Frame List Base Address Register (PERIODICLISTBASE)," for more information.

Address: 0xFC0B_0154 (DEVICEADDR)   Access: User read/write

Figure 21-20. Device Address Register (DEVICEADDR)

Table 21-23. DEVICEADDR Field Descriptions

Field	Description
31–25 USBADR	Device Address. This field corresponds to the USB device address.
24–0	Reserved, must be cleared.

21.3.3.7 Current Asynchronous List Address Register (ASYNCLISTADDR)

The ASYNCLISTADDR register contains the address of the next asynchronous queue head to executed by the host.

On the USB OTG module, 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 21.3.3.8, "Endpoint List Address Register (EPLISTADDR)," for more information.

Address: 0xFC0B_0158 (ASYNCLISTADDR)   Access: User read/write

Figure 21-21. Current Asynchronous List Address Register (ASYNCLISTADDR)

MCF5329 Reference Manual, Rev 3

Table 21-24. 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.

21.3.3.8 Endpoint List Address Register (EPLISTADDR)

This register is not defined in the EHCI specification. For the USB OTG module in 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.

On the USB OTG module, 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 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
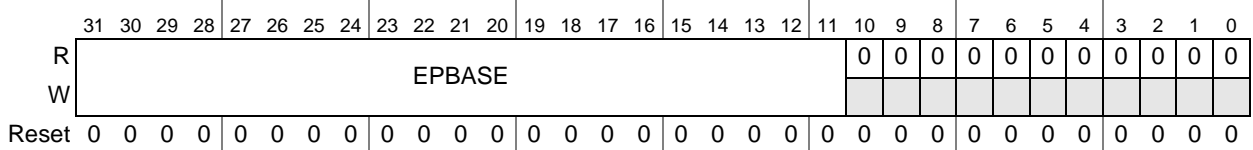
Figure 21-22. Endpoint List Address Register (EPLISTADDR)

Table 21-25. 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.

21.3.3.9 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
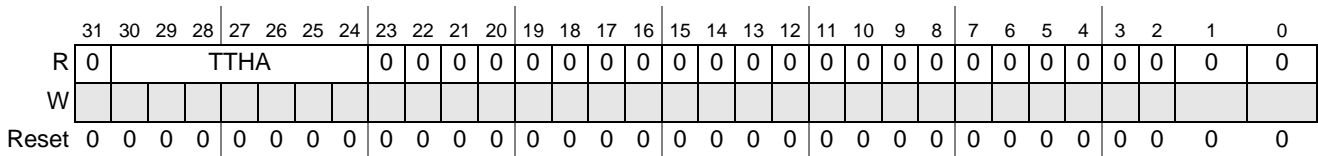
Figure 21-23. Host TT Asynchronous Buffer Control (TTCTRL)

Table 21-26. 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.

21.3.3.10 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 21-24. Master Interface Data Burst Size (BURSTSIZE)

Table 21-27. 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.

21.3.3.11 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
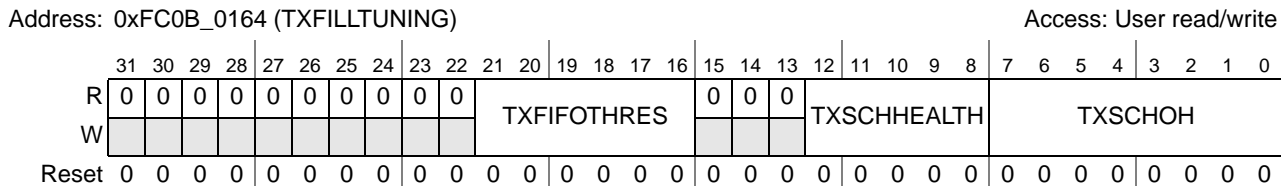
Figure 21-25. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 21-28. 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 21-28. 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: $$\frac{TXFIFOTHRES \times (BURSTSIZE \times 4)}{40 \times TimeUnit}$$ **Eqn. 21-1** 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.

21.3.3.12 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
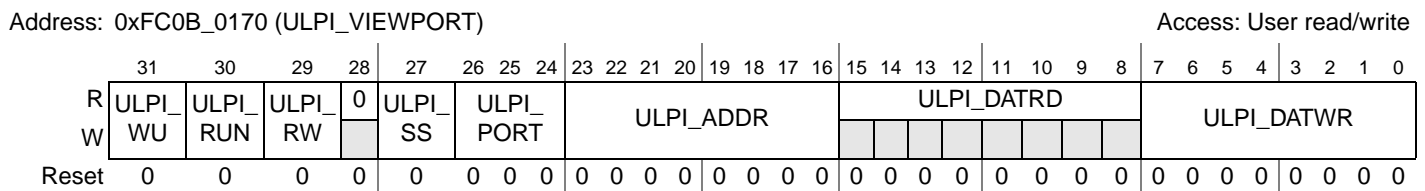
Figure 21-26. ULPI Register Access (ULPI VIEWPORT)

MCF5329 Reference Manual, Rev 3

Table 21-29. 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.

21.3.3.13 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 21-27. Configure Flag Register (CONFIGFLAG)

Table 21-30. CONFIGFLAG Field Descriptions

Field	Description
31–0	Reserved. (0x0000_0001, all port routings default to this host)

21.3.3.14 Port Status and Control Registers (PORTSC*n*)

Both USB modules contain 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 21-28. Port Status and Control Register (PORTSC1*)*

MCF5329 Reference Manual, Rev 3

Table 21-31. PORTSC1 Field Descriptions

Field	Description
31–30 PTS	Port transceiver select. Controls which parallel transceiver interface is selected. This field is read-only for the host and is set to 11 to indicate a FS/LS on-chip transceiver. The following settings apply for the OTG module: 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 (USB host and USB OTG): 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 (USB OTG only): 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 (USB OTG-only). 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 (USB OTG-only). In host mode, this can work with an external power control circuit.

Table 21-31. 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. Controls the link indicator signals and is valid for host mode only. Refers to the *USB Specification Revision 2.0* for a description on how these bits are used. 00 Off 01 Amber 10 Green 11 Undefined This field is output from the module on the USB port control signals for use by an external LED driving circuit. For this device, the port indicator signals are implemented as status bits within the CCM.
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 (USB host and USB OTG):
	   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 (USB OTG only):
	   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 21-31. PORTSC1 Field Descriptions (continued)

Field	Description
7 SUSP	Suspend 0 Port not in suspend state. 1 Port in suspend state. Host mode (USB host and USB OTG): The PE and SUSP bits define the port state as follows: | PE	SUSP	Port State
---	---	---
0	*x*	Disable
1	0	Enable

1	1	Suspend	
			
			    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 (USB OTG only):
			    In device mode, this bit is a read-only status bit.

6
FPR
	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 (USB host and USB OTG):
	    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 *USB Specification Revision 2.0*. 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 (USB OTG only):
	    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.

5 OCC	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*n*_PWRFAULT signal for this condition. For device-only implementations (USB OTG only), this bit must always be cleared. 0 No over-current. 1 Over-current detect.
4 OCA	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*n*_PWRFAULT signal for this condition. For device-only implementations (USB OTG only), this bit must always be cleared. 0 Port not in over-current condition. 1 Port currently in over-current condition.

Table 21-31. 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 (USB OTG only), 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 (USB host and USB OTG): 　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 (USB USB OTG only): 　The device port is always enabled. (This bit is set).
1 CSC	Connect change status. Host mode (USB host and USB OTG): 　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 (USB USB OTG only), 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)

21.3.3.15 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		DPIE	1MSE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE		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											IDPU	DP	OT		VC	VD
Reset	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Figure 21-29. On-the-Go Status and Control Register (OTGSC)

Table 21-32. 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 21-32. 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 21-32. 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.

21.3.3.16 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 21-30. USB Mode Register (USBMODE)

Table 21-33. USBMODE Field Descriptions

Field	Description
31–5	Reserved, must be cleared.

4
SDIS
	Stream disable.
	0 Inactive.
	1 Active.
	Host mode (USB host and USB OTG):
	    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 (USB OTG only):
	    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 21.5.3.4.4, "Control Endpoint Operation." 0 Setup lockouts on. 1 Setup lockouts off (software requires use of the USBCMD[SUTW] bit).

Table 21-33. 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 (default for the USB host module) **Note:** The USB OTG module must be initialized to the desired operating mode after reset.

21.3.3.17 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 by the USB OTG module in device mode.

Address: 0xFC0B_01AC (EPSETUPSR)   Access: User read/write

Figure 21-31. Endpoint Setup Status Register (EPSETUPSR)

Table 21-34. 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.

21.3.3.18 Endpoint Initialization Register (EPPRIME)

This register is not defined in the EHCI specification. This register is used to initialize endpoints and is used only by the USB OTG module in device mode.

Address: 0xFC0B_01B0 (EPPRIME)   Access: User read/write

Figure 21-32. Endpoint Initialization Register (EPPRIME)

MCF5329 Reference Manual, Rev 3

Table 21-35. 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.

21.3.3.19 Endpoint Flush Register (EPFLUSH)

This register is not defined in the EHCI specification. This register used only by the USB OTG module in device mode.

Address: 0xFC0B_01B4 (EPFLUSH)   Access: User read/write

Figure 21-33. Endpoint Flush Register (EPFLUSH)

Table 21-36. 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.

21.3.3.20 Endpoint Status Register (EPSR)

This register is not defined in the EHCI specification. This register is only used by the USB OTG module 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 21-34. Endpoint Status Register (EPSR)

Table 21-37. 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.

21.3.3.21 Endpoint Complete Register (EPCOMPLETE)

This register is not defined in the EHCI specification. This register is used only by the USB OTG module 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 21-35. Endpoint Complete Register (EPCOMPLETE)

Table 21-38. 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 21-38. 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.

21.3.3.22 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 21-36. Endpoint Control 0 (EPCR0)

Table 21-39. 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 21-39. 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

21.3.3.23 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)　　　　　　　　　　　　　　　　　　　　　　Access: User read/write
　　　　　　0xFC0B_01C8 (EPCR2)
　　　　　　0xFC0B_01CA (EPCR3)

	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 21-37. Endpoint Control Registers (EPCR*n*)

Table 21-40. 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 21-40. 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

21.4 Functional Description

Each module (USB host and USB OTG) can be broken down into functional sub-blocks as described below.

21.4.1 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.

21.4.2 DMA Engine

Both USB modules contain local DMA engines. 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 (USB OTG module only), 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.

21.4.3 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 (USB OTG module only), multiple FIFO channels maintain for each of the active endpoints in the system.

In host mode, the USB host and 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.

21.4.4 Physical Layer (PHY) Interface

Readers should familiarize themselves with chapter 7 of the *Universal Serial Bus Specification, Revision 2.0.* The USB host and 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 modules only support certain combinations of PHY interfaces and USB functionality. Refer to the Table 21-41 for more information.

Table 21-41. USB Network Speed and Required Physical Interface

USB Mode and Speed	DP and DN On-Chip Analog XCVR Active	I^{2}C	FEC	External Integrated Circuit Required
USB Host FS/LS	Yes	No	Yes	See Section 21.4.4.1, "USB On-Chip Transceiver Required External Components"
USB Device FS	Yes	No	Yes	See Section 21.4.4.1, "USB On-Chip Transceiver Required External Components"
Host/Device ULPI HS/FS	No	Yes	No	Maxim

21.4.4.1   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.

Further, host operation requires 15k Ω external resistors to be connected from DP and DM ports to ground. Device operation requires a 1.5k Ω pull-up resistor on DP (full-speed operation). The other end of this pull-up resistor should be controlled by an open-drain, low impedence output pin with a 3.3V I/O power supply. Readers are directed to section 7.1.5.1 of the USB 2.0 specification.

21.5   Initialization/Application Information

21.5.1   Host Operation

Enhanced Host Controller Interface (EHCI) Specification defines the general operational model for the USB modules 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.

21.5.1.1   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 21-4.

MCF5329 Reference Manual, Rev 3

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.

21.5.2 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 21-38, 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.
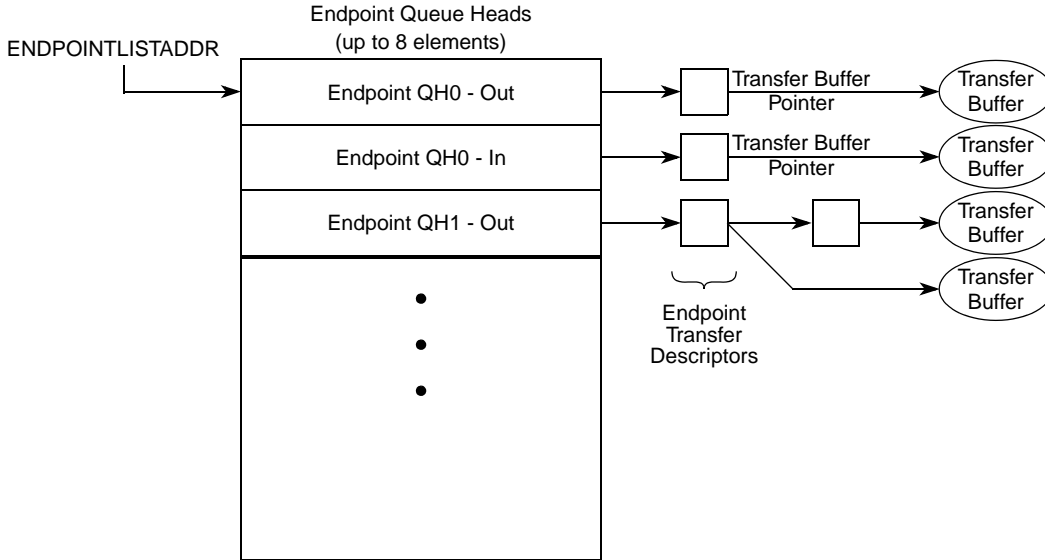
Figure 21-38. End Point Queue Head Organization

21.5.2.1   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 21-39 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 21-39. Endpoint Queue Head Layout

[1] Offsets 0x08 through 0x20 contain the transfer overlay.

21.5.2.1.1 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 21-42. 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 21-42. 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.

21.5.2.1.2   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 21-43. 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.

21.5.2.1.3   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 21.5.2.2, "Endpoint Transfer Descriptor (dTD)," for a description of the overlay fields.

21.5.2.1.4 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 21.5.3.4.4, "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 21-44. 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.

21.5.2.2 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 21.5.3.6, "Managing Transfers with Transfer Descriptors."

Figure 21-40. Endpoint Transfer Descriptor (dTD)

21.5.2.2.1 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 21-45. 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.

21.5.2.2.2   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 21-46. 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 21-46. dTD Token (continued)

Field	Description
11–10 MultO	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.
9–8	Reserved. Reserved for future use and must be cleared.
7–0 Status	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:

Bit	Status Field Description
7	Active. Set by software to enable the execution of transactions by the device controller.
6	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.
5	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).
4	Reserved.
3	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).
2–0	Reserved.

21.5.2.2.3 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 21-47. 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.

21.5.3 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.

21.5.3.1 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 21.3.3.3, "USB Interrupt Enable Register (USBINTR)," and Section 21.3.3.2, "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 21.5.2.1, "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 21.5.3.2.1, "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.

21.5.3.2   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 21-41 depicts the state of a USB 2.0 device.
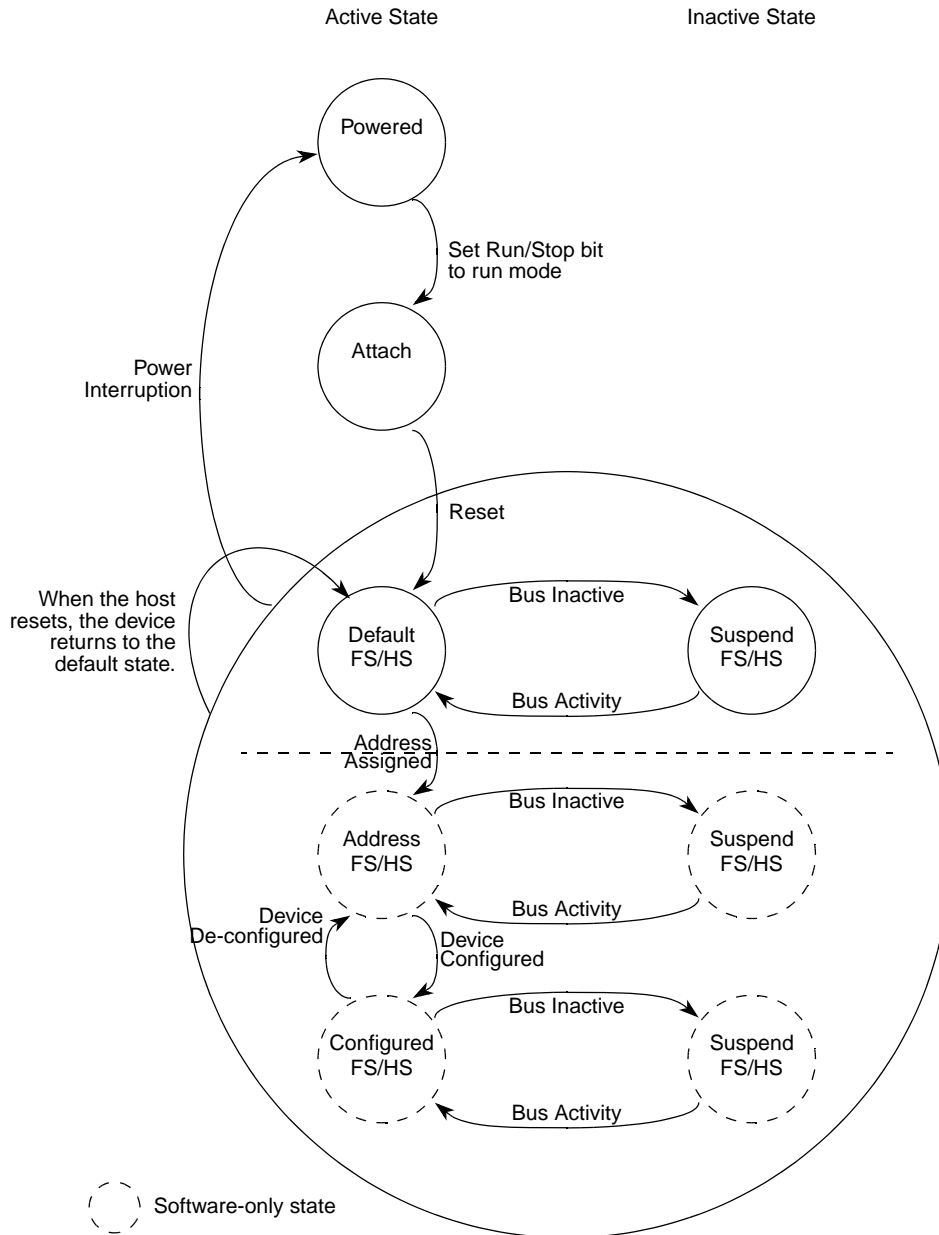
Figure 21-41. USB 2.0 Device States

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 21-48. Device Controller State Information Bits

Bit	Register
DC Suspend (SLI)	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.

21.5.3.2.1   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.

21.5.3.2.2 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.

21.5.3.3   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.

21.5.3.3.1   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 21-49 shows how to construct a configuration word for endpoint initialization.

Table 21-49. 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

21.5.3.3.2 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 21-50. 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

21.5.3.3.3 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.

21.5.3.4 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.

21.5.3.4.1 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.

21.5.3.4.2 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.

21.5.3.4.3 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 21-51 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 21.5.2.1.1, "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 21-51. 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 21-52. 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 21-53. 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.

21.5.3.4.4   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 21-54 shows the device controller response to packets on a control endpoint according to the device controller state.

Table 21-54. 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.

21.5.3.4.5 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 21-55. 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 21-55. 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

21.5.3.5   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 21-42). 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 21.5.3.6.1, "Software Link Pointers").
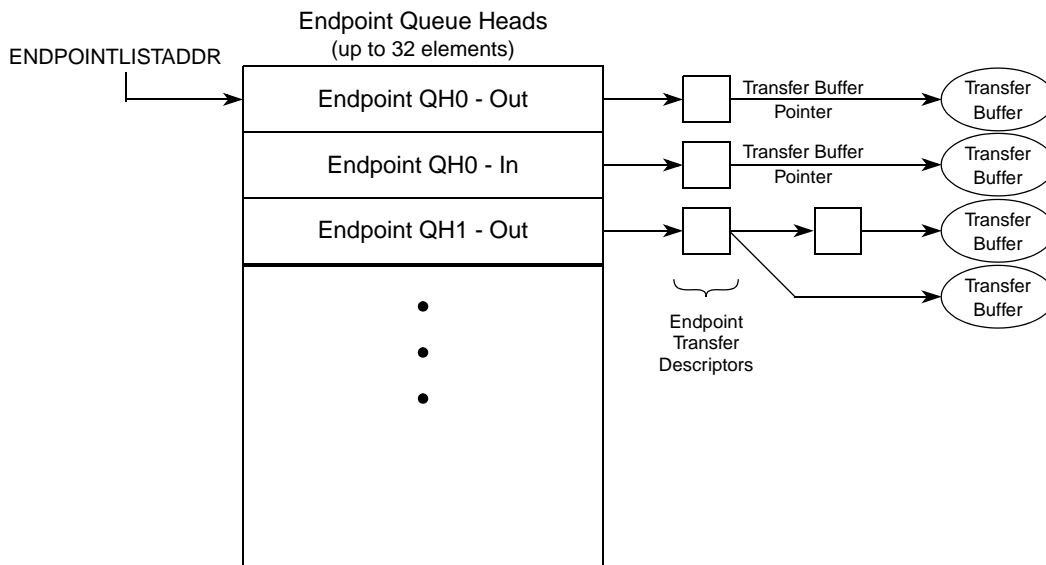
Figure 21-42. Endpoint Queue Head Diagram

In addition to current and next pointers and the dTD overlay examined in Section 21.5.3.4, "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.

21.5.3.5.1 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.

21.5.3.5.2 Setup Transfers Operation

As discussed in Section 21.5.3.4.4, "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 21.5.3.6.5, "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.

21.5.3.6 Managing Transfers with Transfer Descriptors

21.5.3.6.1 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.
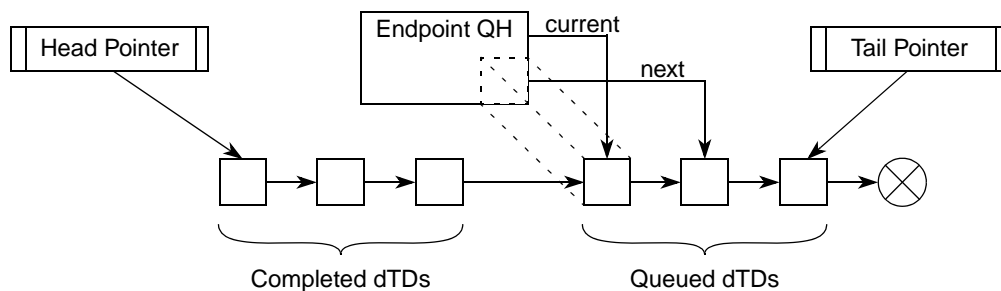
Figure 21-43. Software Link Pointers

NOTE

Check the status of each dTD to determine completed status.

21.5.3.6.2 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.

21.5.3.6.3   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.

21.5.3.6.4   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.

<p align="center">**CAUTION**</p>

> 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 21.5.3.6.6, "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.

21.5.3.6.5 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.

21.5.3.6.6 Device Error Matrix

The following table summarizes packet errors not automatically managed by the USB OTG module.

Table 21-56. 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 21-57. 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.

21.5.4 Servicing Interrupts

The interrupt service routine must understand there are high frequency, low frequency, and error operations to order accordingly.

21.5.4.1 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 21-58. 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 21.5.3.5, "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 21.5.3.5, "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.

21.5.4.1.1 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 21-59. 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.

21.5.4.1.2   Error Interrupts

Error interrupts are least frequent and should be placed last in the interrupt service routine.

Table 21-60. 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.

21.5.5   Deviations from the EHCI Specifications

The host mode operation of the USB host and OTG modules 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 (USB host and OTG modules)—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation (USB OTG module only)—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 modules do 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.

21.5.5.1   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.

21.5.5.1.1   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 21.3.2.3, "Host Controller Structural Parameters Register (HCSPARAMS)" for usage information.

21.5.5.1.2 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.

21.5.5.1.3 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 21-61.

Table 21-61. 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)]

21.5.5.1.4 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 PORTSCn[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.

21.5.5.1.5 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 slightly 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 21-62 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 21-62. 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.

21.5.5.2   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.

21.5.5.3   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).

21.5.5.4   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 21.3.3.2, "USB Status Register (USBSTS)," and Section 21.3.3.3, "USB Interrupt Enable Register (USBINTR)," for more information.

21.5.5.5   Embedded Design

This is an embedded USB host controller as defined by the EHCI specification; therefore, it does not implement the PCI configuration registers.

21.5.5.5.1   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.

21.5.5.6   Miscellaneous Variations from EHCI

21.5.5.6.1   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 PORTSCn register providing a capability not defined by the EHCI specification.

21.5.5.6.2   Discovery

Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the PORTSCn 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 PORTSCn[PR] bit to reset the device.
- Software shall clear the PORTSCn[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, PORTSCn[PO] bit is read-only and always reads 0.
- A 2-bit port speed indicator field has been added to PORTSCn 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 PORTSCn 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 22
Liquid Crystal Display Controller (LCDC)

22.1 Introduction

The liquid crystal display controller (LCDC) provides display data for external gray-scale or color LCD panels. The LCDC is capable of supporting black-and-white, gray-scale, passive-matrix color (passive color or CSTN), and active-matrix color (active color or TFT) LCD panels.

22.1.1 Block Diagram

The LCD controller block diagram is shown below.
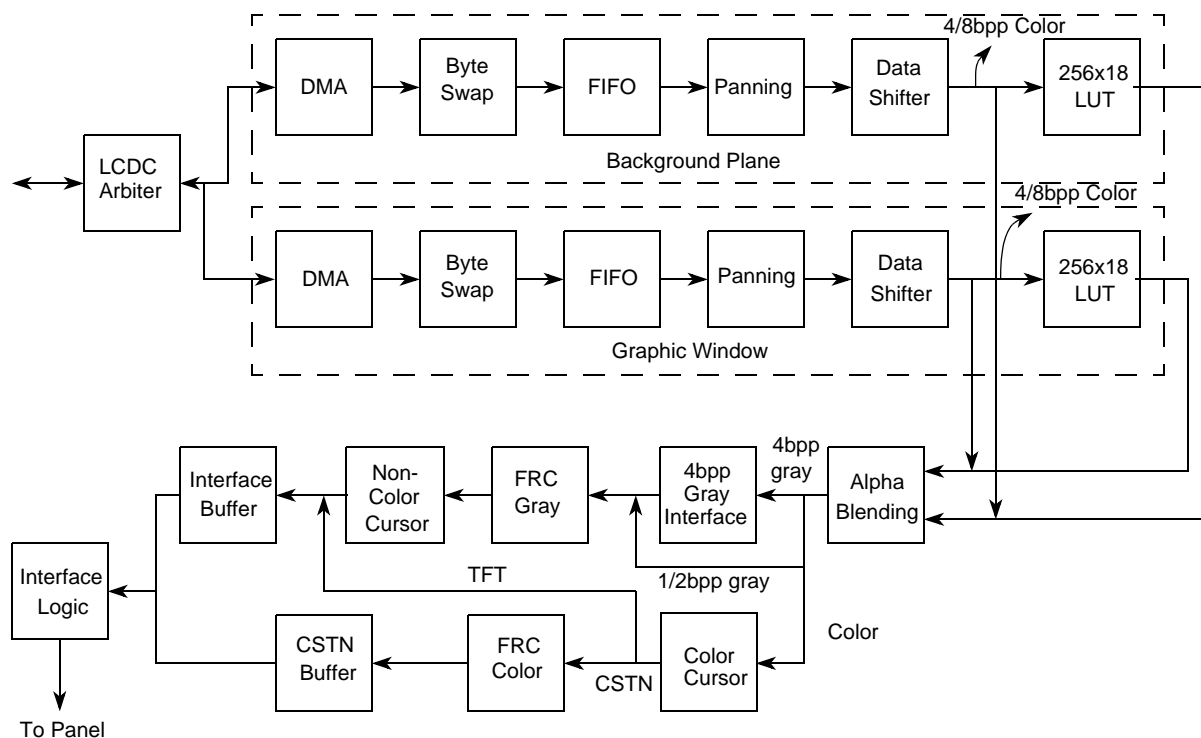
Figure 22-1. LCDC Block Diagram

22.1.2 Features

The LCDC provides the following features:

- Support for single (non-split) screen monochrome or color LCD panels and self-refresh type LCD panels

- 16 simultaneous gray-scale levels from a palette of 16 for monochrome display
- Support for:
  - Maximum resolution of 800×600
  - Passive color panel:
    - 4 (mapped to RGB444) / 8 (mapped to RGB444) / 12 (RGB444) bits per pixel (bpp)
  - TFT panel:
    - 4 (mapped to RGB666) / 8 (mapped to RGB666) / 12 (RGB444) / 16 (RGB565) / 18 (RGB666) bpp
  - 16 and 256 colors out of a palette of 4096 colors for 4 bpp and 8 bpp CSTN display respectively
  - 16 and 256 colors out of a palette of 256K colors for 4 bpp and 8 bpp TFT display respectively
  - True 4096 colors for a 12 bpp display
  - True 64K colors for 16 bpp
  - True 256K colors for 18 bpp
  - Additional support details are shown in Table 22-1

Table 22-1. Supported Panel Characteristics

Panel Type	Bit/Pixel	Panel Interface (Bits)	Number of Gray Level/Color
Monochrome	1	1, 2, 4, 8	black-and-white
	2	1, 2, 4, 8	4 out of palette of 16
	4	1, 2, 4, 8	16
CSTN	4, 8	12	16, 256 out of palette of 4096
	12	12	4096
TFT	4, 8	18	16, 256 out of palette of 256K
	12, 16, 18	12, 16, 18	4096, 64K, 256K

- Standard panel interface for common LCD drivers
- Panel interface of 1-, 2-, 4-, 8-bit for monochrome panels
- Panel interface of 12-, 16-, 18-bit for color panels
- For 4 bpp and 8 bpp a palette table is used for re-mapping of data from memory, independent of type of panel used. For the 1 bpp, 2 bpp, 12 bpp, 16 bpp, and 18 bpp the palette table is by-passed.
- Interface to passive and active color panel (TFT)
- Supports timing requirements for Sharp 240×320 HR-TFT panel
- Hardware-generated cursor with blink, color, and size programmability
- Logical operation between color hardware cursor and background
- Hardware panning (soft horizontal scrolling)
- 8-bit pulse-width modulator for software contrast control
- Graphic window support for viewfinder function in color display

- Graphic window color keying for graphical hardware cursor
- 256 transparency levels for alpha blending between graphic window and background plane

22.2 External Signal Description

Table 22-2 describes the LCD controller signals.

Table 22-2. LCD Signals

Signal	I/O	Function
LCD_D[17:0]	O	Line data. LCD data bus.
LCD_FLM/ LCD_VSYNC	O	Passive matrix: First line marker Active matrix: Vertical sync pulse. Indicates start of next frame.
LCD_LP/ LCD_HSYNC	O	Passive matrix: Line pulse Active matrix: Horizontal sync pulse. Indicates start of next line.
LCD_LSCLK	O	Shift clock. Clock for latching data into the display driver's internal shift register.
LCD_ACD/ LCD_OE	O	Passive matrix: Alternate crystal direction Active matrix: Output enable to enable data to be shifted onto the display.
LCD_CONTRAST	O	Contrast. Controls the LCD bias voltage for contrast control.
LCD_PS	O	Power save. Controls signal output for source driver (Sharp HR-TFT 240x320 panels only)
LCD_CLS	O	Gate driver clock signal. Start signal output for gate driver, inverted version of LCD_PS (Sharp HR-TFT 240x320 panels only).
LCD_REV	O	Reverse control. Signal for common electrode driving signal preparation (Sharp HR-TFT 240x320 panels only).
LCD_SPL_SPR	O	Sampling start signal. Sets the horizontal scan direction (Sharp HR-TFT 240x320 panels only).

22.3 Memory Map/Register Definition

The LCDC memory space contains 21 32-bit registers for display parameters, a read-only status register, and two 256 × 18 color-mapping RAMs—one for graphic window and the other for background plane. The color-mapping RAMs are physically located inside the palette lookup table module.

Table 22-3 summarizes these registers and their addresses. Only longword access is supported. Byte and word access is undefined.

Table 22-3. LCD Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_C000	Screen Start Address Register (LCD_SSAR)	32	R/W	0x0000_0000	22.3.1/22-5
0xFC0A_C004	LCD Size Register (LCD_SR)	32	R/W	0x0000_0000	22.3.2/22-5
0xFC0A_C008	LCD Virtual Page Width Register (LCD_VPW)	32	R/W	0x0000_0000	22.3.3/22-5
0xFC0A_C00C	LCD Cursor Position Register (LCD_CPR)	32	R/W	0x0000_0000	22.3.4/22-6

Table 22-3. LCD Controller Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_C010	LCD Cursor Width Height and Blink Register (LCD_CWHB)	32	R/W	0x0101_00FF	22.3.5/22-7
0xFC0A_C014	LCD Color Cursor Mapping Register (LCD_CCMR)	32	R/W	0x0000_0000	22.3.6/22-8
0xFC0A_C018	LCD Panel Configuration Register (LCD_PCR)	32	R/W	0x0000_0000	22.3.7/22-9
0xFC0A_C01C	LCD Horizontal Configuration Register (LCD_HCR)	32	R/W	0x0400_0000	22.3.8/22-12
0xFC0A_C020	LCD Vertical Configuration Register (LCD_VCR)	32	R/W	0x0400_0000	22.3.9/22-12
0xFC0A_C024	LCD Panning Offset Register (LCD_POR)	32	R/W	0x0000_0000	22.3.10/22-13
0xFC0A_C028	LCD Sharp Configuration Register (LCD_SCR)	32	R/W	0x400C_0373	22.3.11/22-14
0xFC0A_C02C	LCD PWM Contrast Control Register (LCD_PCCR)	32	R/W	0x0000_0000	22.3.12/22-16
0xFC0A_C030	LCD DMA Control Register (LCD_DCR)	32	R/W	0x8010_0004	22.3.13/22-16
0xFC0A_C034	LCD Refresh Mode Control Register (LCD_RMCR)	32	R/W	0x0000_0000	22.3.14/22-17
0xFC0A_C038	LCD Interrupt Configuration Register (LCD_ICR)	32	R/W	0x0000_0000	22.3.15/22-18
0xFC0A_C03C	LCD Interrupt Enable Register (LCD_IER)	32	R/W	0x0000_0000	22.3.16/22-19
0xFC0A_C040	LCD Interrupt Status Register (LCD_ISR)	32	R/W	0x0000_0000	22.3.17/22-20
0xFC0A_C050	LCD Graphic Window Start Address Register (LCD_GWSAR)	32	R/W	0x0000_0000	22.3.18/22-22
0xFC0A_C054	LCD Graphic Window Size Register (LCD_GWSR)	32	R/W	0x0000_0000	22.3.19/22-22
0xFC0A_C058	LCD Graphic Window Virtual Page Width Register (LCD_GWVPW)	32	R/W	0x0000_0000	22.3.20/22-23
0xFC0A_C05C	LCD Graphic Window Panning Offset Register (LCD_GWPOR)	32	R/W	0x0000_0000	22.3.21/22-23
0xFC0A_C060	LCD Graphic Window Position Register (LCD_GWPR)	32	R/W	0x0000_0000	22.3.22/22-24
0xFC0A_C064	LCD Graphic Window Control Register (LCD_GWCR)	32	R/W	0x0000_0000	22.3.23/22-24
0xFC0A_C068	LCD Graphic Window DMA Control Register (LCD_GWDCR)	32	R/W	0x8010_0004	22.3.24/22-26
0xFC0A_C800 ... 0xFC0A_CBFC	Background Look-up Table (BGLUT)	32	R/W	—	22.3.25/22-26
0xFC0A_CC00 ... 0xFC0A_CFFC	Graphic Window Look-up Table (GWLUT)	32	R/W	—	22.3.25/22-26

22.3.1 LCDC Screen Start Address Register (LCD_SSAR)

The screen start address register specifies the start address of the LCD screen.

Address: 0xFC0A_C000 (LCD_SSAR) 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					SSA				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
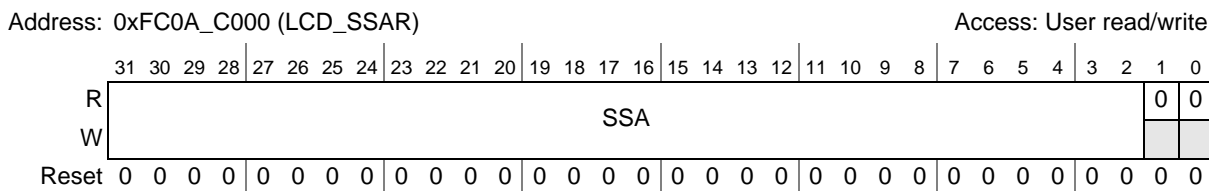
Figure 22-2. LCD Screen Start Address Register (LCD_SSAR)

Table 22-4. LCD_SSAR Field Descriptions

Field	Description
31–2 SSA	Screen start address of the LCD panel. Holds pixel data for a new frame from the SSA address. This field must start at a location that enables a complete picture to be stored in a 4 Mbyte memory boundary (A [21:0]). A [31:22] has a fixed value for a picture's image.
1–0	Reserved, must be cleared.

22.3.2 LCDC Size Register (LCD_SR)

The size register defines the height and width of the LCD screen.

Address: 0xFC0A_C004 (LCD_SR) 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		XMAX	0 0 0 0	0 0 0 0	0 0 0	YMAX	
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
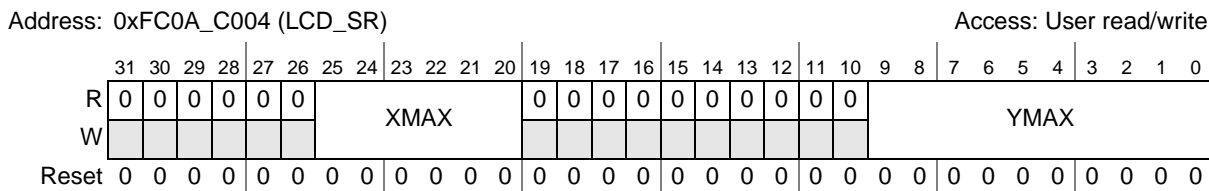
Figure 22-3. LCD Size Register (LCD_SR)

Table 22-5. LCD_SR Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25–20 XMAX	Screen width divided by 16. Holds screen x-axis size in pixels, divided by 16. For black-and-white panels (1 bpp), XMAX[0] is ignored, forcing the x-axis of the screen size to be a multiple of 32 pixels/line. A value of zero in this bit field is reserved. **Note:** The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x32.
19–10	Reserved, must be cleared.
9–0 YMAX	Screen height. Specifies the height of the LCD panel in terms of pixels or lines. The lines are numbered from 1 to YMAX for a total of YMAX lines. A value of zero in this bit field is reserved. **Note:** The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x258.

22.3.3 LCDC Virtual Page Width Register (LCD_VPW)

The virtual page width register defines the width of the virtual page for the LCD panel.

Address: 0xFC0A_C008 (LCD_VPW)   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			VPW							
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
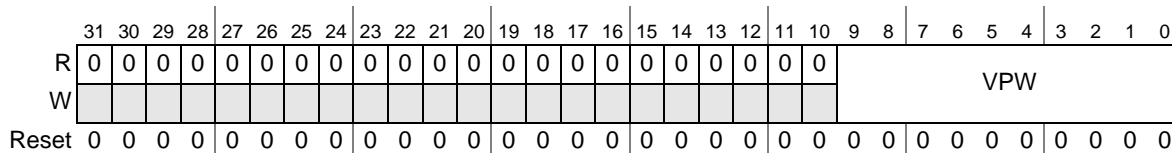
Figure 22-4. LCD Virtual Page Width Register (LCD_VPW)

Table 22-6. LCD_VPW Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9–0 VPW	Virtual page width. Defines the virtual page width of the LCD panel. The VPW bits represent the number of 32-bit longwords required to hold the data for one virtual line. VPW is used in calculating the starting address representing the beginning of each displayed line.

22.3.4 LCDC Cursor Position Register (LCD_CPR)

The cursor position register is used to determine the starting position of the cursor on the LCD panel.

Address: 0xFC0A_C00C (LCD_CPR)   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	CC		0	OP	0	0		CXP									0	0	0	0	0	0			CYP							
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
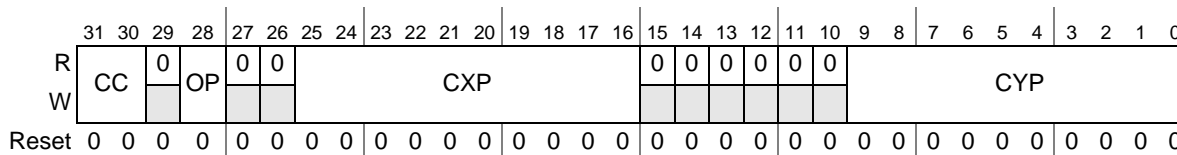
Figure 22-5. LCD Cursor Position Register (LCD_CPR)

Table 22-7. LCD_CPR Field Descriptions

Field	Description
31–30 CC	Cursor control. Controls the format of the cursor and the type of arithmetic operations, if enabled by the OP bit. { OP/CC/Cursor Format table below }
29	Reserved, must be cleared.
28 OP	Arithmetic operation control. Enables/disables arithmetic operations between the background and the cursor. 0 Arithmetic operation disabled. 1 Arithmetic operation enabled. (Should only be used for color modes.)
27–26	Reserved, must be cleared.
25–16 CXP	Cursor X-position. Indicates the cursor's horizontal starting position in pixel count (from 0 to LCD_SR[XMAX]).
15–10	Reserved, must be cleared.
9–0 CYP	Cursor Y-position. Indicates the cursor's vertical starting position in pixel count (from 0 to LCD_SR[YMAX]).

OP[1]	CC	Cursor Format
X	00	Transparent, cursor is disabled
0	01	1 for non-color displays; color defined in the LCD_CCMR register for color displays
0	10	Reversed, INV background for non-color displays; INV color defined in the LCD_CCMR register for color displays
0	11	0 for non-color displays; AND between background and cursor for color displays
1	01	OR between background and cursor
1	10	XOR between background and cursor
1	11	AND between background and cursor

[1] OP = 1 for color modes only.

22.3.5 LCDC Cursor Width Height and Blink Register (LCD_CWHB)

The cursor width height and blink register is used to determine the width and height of the cursor, and how it blinks.

Address: 0xFC0A_C010 (LCD_CWHB)   Access: User read/write
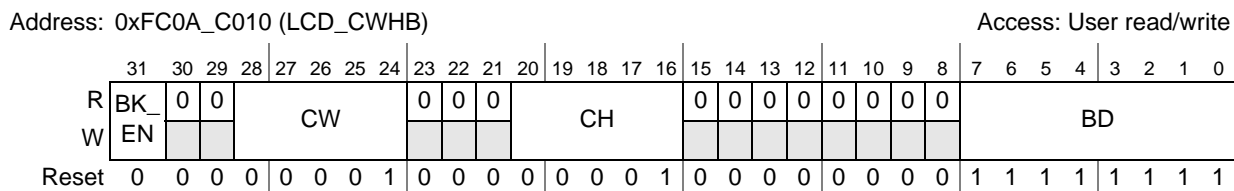
Figure 22-6. LCD Cursor Width Height and Blink Register (LCD_CWHB)

Table 22-8. LCD_CWHB Field Descriptions

Field	Description
31 BK_EN	Blink enable. Determines whether the cursor blinks or remains steady. 0 Blink is disabled. 1 Blink is enabled.
30–29	Reserved, must be cleared.
28–24 CW	Cursor width. Specifies the width of the hardware cursor in pixels. This field can be any value between 1 and 31. Setting this field to zero disables the cursor.
23–21	Reserved, must be cleared.
20–16 CH	Cursor height. Specifies the height of the hardware cursor in pixels. This field can be any value between 1 and 31. Setting this field to zero disables the cursor.
15–8	Reserved, must be cleared.
7–0 BD	Blink divisor. Sets the cursor blink rate. A 32 Hz clock from the real-time clock (RTC) module is used to clock the 8-bit up counter. When the counter value equals BD, the cursor toggles on/off. Hence the larger the BD, the slower the cursor is blinking. The fastest cursor blinking rate is when BD is 0.

22.3.6 LCDC Color Cursor Mapping Register (LCD_CCMR)

The color cursor mapping register defines the color of the cursor in passive or TFT color modes. If the bpp mode setting is smaller than 18bpp, the cursor color component bits should be put in the msbs. For example, if the color cursor of RGB = (0x1A, 0x26, 0x05) is wanted, the CUR_COL_R, CUR_COL_G and CUR_COL_B should be set to 0x34, 0x26 and 0x0A respectively.

Address: 0xFC0A_C014 (LCD_CCMR)   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	CUR_COL_R	CUR_COL_G	CUR_COL_B
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
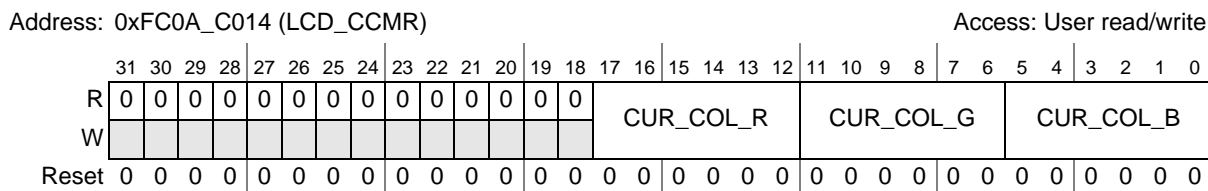
Figure 22-7. LCD Color Cursor Mapping Register (LCD_CCMR)

Table 22-9. LCD_CCMR Field Descriptions

Field	Description
30–18	Reserved, must be cleared.
17–12 CUR_COL_R	Cursor red field. Defines the red component of the cursor color in color mode. 0x00 No red. ... 0x3F Full red.

Table 22-9. LCD_CCMR Field Descriptions (continued)

Field	Description
11–6 CUR_COL_G	Cursor green field. Defines the green component of the cursor color in color mode. 0x00 No green. ... 0x3F Full green.
5–0 CUR_COL_B	Cursor blue field. Defines the blue component of the cursor color in color mode. 0x00 No blue. ... 0x3F Full blue.

22.3.7 LCDC Panel Configuration Register (LCD_PCR)

The panel configuration register defines all of the properties of the LCD screen.

Address: 0xFC0A_C018 (LCD_PCR)   Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	TFT	COL OR	PBSIZ		BPIX			PIX POL	FLM POL	LP POL	CLK POL	OE POL	SCLK IDLE	END_ SEL	SWAP_ SEL	REV_ VS
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	ACD SEL	ACD							SCLK SEL	SHARP	PCD					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-8. LCD Panel Configuration Register (LCD_PCR)

Table 22-10. LCD_PCR Field Descriptions

Field	Description
31 TFT	TFT display enable. Controls the format and timing of the output control signals. Active and passive displays use different signal timing formats as described in Section 22.4.9, "Panel Interface Signals and Timing." TFT also controls the use of the frame rate control (FRC) in color mode. Refer to below table for TFT/COLOR setting usage. 0 The LCD panel is a passive display. 1 The LCD panel is an active display: digital CRT signal format, FRC is bypassed. <table><tr><th>TFT</th><th>COLOR</th><th>LCD Display</th></tr><tr><td>0</td><td>0</td><td>Monochrome</td></tr><tr><td>0</td><td>1</td><td>CSTN</td></tr><tr><td>1</td><td>0</td><td>Reserved</td></tr><tr><td>1</td><td>1</td><td>TFT</td></tr></table>
30 COLOR	Color display enable. Activates three channels of FRC in passive mode to allow use of the special 2 2/3 pixels per output vector format. Refer to TFT bit field description for TFT/COLOR setting usage. 0 The LCD panel is a monochrome display. 1 The LCD panel is a color display.
29–28 PBSIZ	Panel bus width. Specifies the panel bus width. Applicable for monochrome or passive matrix color monitors. For passive color panels, only a 12-bit panel bus width is supported. 00 1-bit 01 Reserved 10 4-bit 11 8-bit
27–25 BPIX	Indicates the number of bits per pixel in memory. 000 1 bpp, FRC bypassed 001 2 bpp 010 4 bpp 011 8 bpp 100 12 bpp (16 bits of memory used) 101 16 bpp 110 18 bpp (32 bits of memory used) 111 Reserved **Note:** To set normal 18 bpp mode, use the following settings: BPIX = 110, END_SEL = 0, SWAP_SEL = X (don't care). To set Microsoft PAL_BGR 18 bpp mode, use the following: BPIX = 110, END_SEL = 1, SWAP_SEL = 1.
24 PIXPOL	Pixel polarity. 0 Active high 1 Active low
23 FLMPOL	First line marker polarity. 0 Active high 1 Active low
22 LPPOL	Line pulse polarity. 0 Active high 1 Active low

Table 22-10. LCD_PCR Field Descriptions (continued)

Field	Description
21 CLKPOL	LCD shift clock polarity. Sets the polarity of the active edge of the LCD shift clock. 0 Active negative edge of LCD_LSCLK (in TFT mode, active on positive edge of LCD_LSCLK) 1 Active positive edge of LCD_LSCLK (in TFT mode, active on negative edge of LCD_LSCLK)
20 OEPOL	LCD output enable polarity. 0 Disable LCD_LSCLK. 1 Enable LCD_LSCLK.
19 SCLKIDLE	LCD SCLK idle enable. Enables/disables LCD_LSCLK when LCD_VSYNC is idle in TFT mode. 0 Disable LCD_LSCLK. 1 Enable LCD_LSCLK.
18 END_SEL	Endian select. Selects the image download into memory as big or little endian format. 0 Little endian. 1 Big endian.
17 SWAP_SEL	Swap Select. LCDC operates in big endian mode internally. Swap Select controls the swap of data before operation in little endian mode. 0 16 bpp, 12 bpp mode. 1 8 bpp, 4 bpp, 2 bpp, 1 bpp mode. **Note:** When SWAP_SEL = 0, byte 3 (bits 31–24), byte 2 (bits 23–16), byte 1 (bits 15–8), byte 0 (bits 7–0) data swapped to byte 1, byte 0, byte 3 and byte 2 respectively. **Note:** When SWAP_SEL = 1, byte 3, byte 2, byte 1, byte 0 data swapped to byte 0, byte 1, byte 2 and byte 3 respectively.
16 REV_VS	Reverse vertical scan. Selects the vertical scan direction as normal or reverse (the image flips along the x-axis). The LCD_SSAR register must be changed accordingly. 0 Vertical scan in normal direction. 1 Vertical scan in reverse direction.
15 ACDSEL	LCD_ACD clock source select. Selects the clock source used by the alternative crystal direction counter. 0 Use LCD_FLM as clock source for ACD count. 1 Use LCD_LP/HSYNC as clock source for ACD count.
14–8 ACD	Alternate crystal direction. Toggles the LCD_ACD signal once every 1–16 LCD_FLM cycles based on the value specified in this field. The actual number of LCD_FLM cycles between toggles is the programmed value plus one. **Note:** For active mode (TFT=1), this parameter is not used.
7 SCLKSEL	LCD_LSCLK select. Selects whether to enable or disable LCD_LSCLK in TFT mode when there is no data output. 0 Disable LCD_OE and LCD_LSCLK in TFT mode when no data output. 1 Always enable LCD_LSCLK in TFT mode even if there is no data output.
6 SHARP	Sharp panel enable. Enables/disables signals for Sharp HR-TFT 240 x 320 panels. 0 Disable Sharp signals. 1 Enable Sharp signals.
5–0 PCD	Pixel clock divider. Indicates the clock divider value. The LCDC clock ($f_{sys/3}$) is divided by N (PCD plus one) to yield the pixel clock rate. Values of 1 to 63 yield N=2 to 64. The pixel clock rate is faster than LCD_LSCLK by a factor equal to the data bus width for monochrome display. For all other displays, the pixel clock rate is the same as LCD_LSCLK. **Note:** Set PCD so that the LCD_LSCLK frequency is less than one-third (TFT mode) or one-fourth (CSTN mode) of the system bus clock ($f_{sys/3}$) frequency. Otherwise, the line data (LCD_D) is incorrect.

22.3.8 LCDC Horizontal Configuration Register (LCD_HCR)

The horizontal configuration register defines the horizontal sync pulse timing. For detailed settings, please refer to the panel's data sheet.

Address: 0xFC0A_C01C (LCD_HCR)   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	H_WIDTH		0 0	0 0 0 0	0 0 0 0	H_WAIT_1		H_WAIT_2	
W									
Reset	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 0 0	0 0 0 0
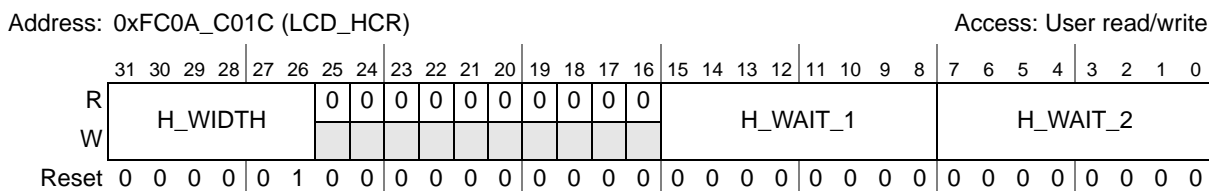
Figure 22-9. LCD Horizontal Configuration Register (LCD_HCR)

Table 22-11. LCD_HCR Field Descriptions

Field	Description
31–26 H_WIDTH	Horizontal sync pulse width. Specifies the number of LCD_LSCLK periods that LCD_HSYNC is activated. The active time is equal to (H_WIDTH + 1) of the LCD_LSCLK periods.
25–16	Reserved, must be cleared.
15–8 H_WAIT_1	Wait between LCD_OE and LCD_HSYNC. In TFT mode, this field specifies the number of LCD_LSCLK periods between the end of LCD_OE signal and the beginning of the LCD_HSYNC. Total delay time equals (H_WAIT_1 + 1) of LCD_LSCLK periods. In CSTN mode, H_WAIT_1 specifies the number of LCD_LSCLK periods between the last display data and the beginning of the LCD_HSYNC signal. Total delay time equals (H_WAIT_1 + 1) of LCD_LSCLK periods.
7–0 H_WAIT_2	Wait between LCD_HSYNC and start of next line. In TFT mode, this field specifies the number of LCD_LSCLK periods between the end of LCD_HSYNC and the beginning of the LCD_OE signal. Total delay time equals (H_WAIT_2 + 3). In CSTN mode, H_WAIT_2 specifies the number of LCD_LSCLK periods between the end of LCD_HSYNC and the first display data in each line. Total delay time equals (H_WAIT_2 + 2) of LCD_LSCLK periods.

22.3.9 LCDC Vertical Configuration Register (LCD_VCR)

The vertical configuration register defines the vertical sync pulse timing. For detailed settings, please refer to the panel's data sheet.

Address: 0xFC0A_C020 (LCD_VCR)   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	V_WIDTH		0 0	0 0 0 0	0 0 0 0	V_WAIT_1		V_WAIT_2	
W									
Reset	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 0 0	0 0 0 0
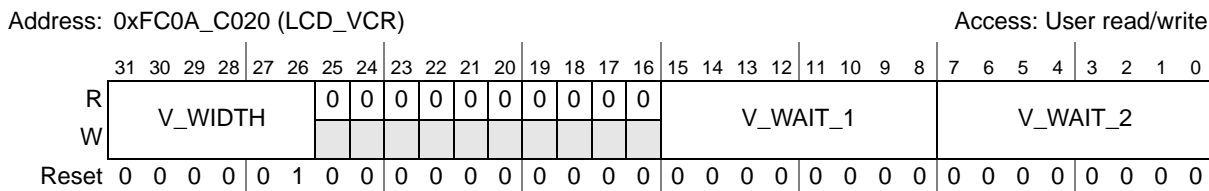
Figure 22-10. LCD Vertical Configuration Register (LCD_VCR)

Table 22-12. LCD_VCR Field Descriptions

Field	Description
31–26 V_WIDTH	Vertical sync pulse width. Specifies the width, in lines, of the LCD_VSYNC pulse for active mode (TFT=1). For a value of 0x01, the vertical sync pulse encompasses one LCD_HSYNC pulse. For a value of 0x02, the vertical sync pulse encompasses two LCD_HSYNC pulses, and so on. For passive mode (TFT=0) and non-color mode, see Figure 22-39.
25–16	Reserved, must be cleared.
15–8 V_WAIT_1	Wait between frames 1. Defines the delay, in lines, between the end of the LCD_OE pulse and the beginning of the LCD_VSYNC pulse for active mode (TFT=1). This field has no meaning in passive non-color mode. The actual delay is V_WAIT_1 lines. In passive color mode, this field is the delay, measured in virtual clock periods, between the last line of the frame to the beginning of the next frame.
7–0 V_WAIT_2	Wait between frames 2. Defines the delay, in lines, between the end of the LCD_VSYNC pulse and the beginning of the LCD_OE pulse of the first line in active mode (TFT=1). The actual delay is V_WAIT_2 lines. Set this field to zero for passive non-color mode. The minimum value of this field is 0x01.

22.3.10 LCDC Panning Offset Register (LCD_POR)

The panning offset register sets up the panning for the image.

Address: 0xFC0A_C024 (LCD_POR)   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	POS			
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 22-11. LCD Panning Offset Register (LCD_POR)

Table 22-13. LCD_POR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4–0 POS	Panning offset. Defines the number of bits that the data from memory is panned to the left before processing. The POS field is read by the LCDC once at the beginning of each frame. For example, in 4 bpp mode, setting POS = 16 shifts the data 16bits, which equates to panning the image by 4 pixels to the left. **Note:** Use the LSSAR register to shift the data more than 32 bits or for 18 bpp panning. To achieve panning of the final image by N bits:

Bits Per Pixel	POS	Effective # of pixels Panned on Image
1	N	N
2	2N	N
4	4N	N
8	8N	N
12/16	16N	N

22.3.11 LCDC Sharp Configuration Register (LCD_SCR)

For 2 bpp modes, full black and full white are the two predefined display levels. The other two intermediate gray-scale shading densities can be adjusted within the Sharp configuration register. The LCD_SCR register also controls the relative delay timing of the LCD_CLS, LCD_REV, and LCD_PS signals. For detailed Sharp panel settings, please refer to panel's data sheet. The TFT timing diagram that shows the relationship between these signals is shown in Figure 22-13.
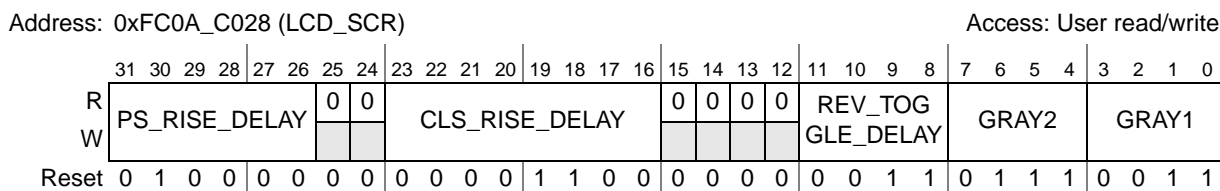
Figure 22-12. LCD Sharp Configuration Register (LCD_SCR)

Table 22-14. LCD_SCR Field Descriptions

Field	Description
31–26 PS_RISE_ DELAY	LCD_PS rise delay. Controls the delay of the rising edge of LCD_PS relative to the falling edge of LCD_CLS. Total delay time is equal to PS_RISE_DELAY LCD_LSCLK periods. 0x00 Zero LCD_LSCLK periods ... 0x3F 63 LCD_LSCLK periods
25–24	Reserved, must be cleared.

Table 22-14. LCD_SCR Field Descriptions (continued)

Field	Description
23–16 CLS_RISE_ DELAY	LCD_CLS rise delay. Controls the delay of the rising edge of LCD_CLS relative to the last line data of the line. Total delay time is equal to (CLS_RISE_DELAY + 1) LCD_LSCLK periods 0x00 1 LCD_LSCLK period … 0xFF 256 LCD_LSCLK periods
15–12	Reserved, must be cleared.
11–8 REV_TOGGLE_ DELAY	LCD_REV toggle delay. Controls the transition delay of LCD_REV relative to the last line data of the line. Total delay time is equal to (REV_TOGGLE_DELAY + 1) LCD_LSCLK periods 0x0 1 LCD_LSCLK period … 0xF 16 LCD_LSCLK periods
7–4 GRAY2	Grayscale 2. Represents one of the two grayscale shading densities. This field is programmable to any value between 0 and 16 (0 and 16 are already defined as two of the four colors).
3–0 GRAY1	Grayscale 1. Represents one of the two grayscale shading densities. This field is programmable to any value between 0 and 16 (0 and 16 are already defined as two of the four colors).
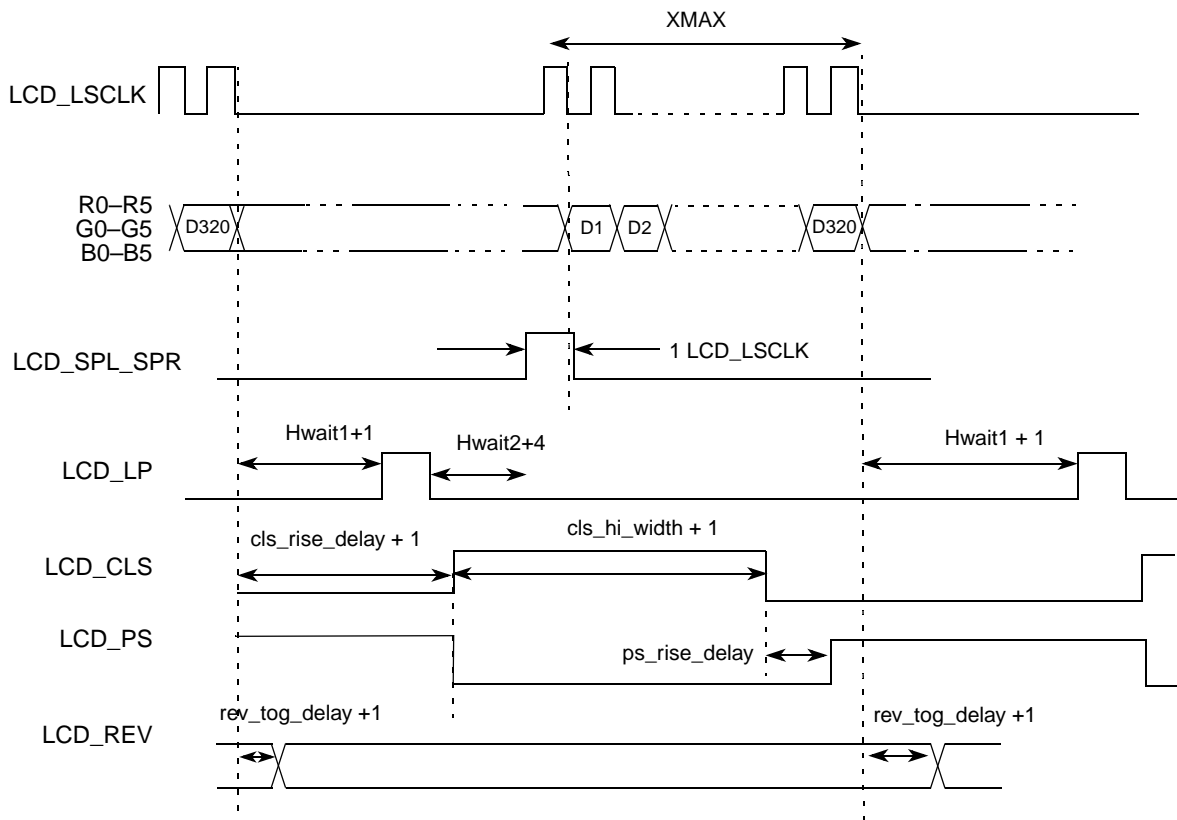
Falling edge of LCD_PS aligns with rising edge of LCD_CLS
The rising edge delay of LCD_PS is programmed by PS_RISE_DELAY
CLS_HI_WIDTH is equal to PWM_SCR0 • 256 + PWM_WIDTH in units of LCD_LSCLK.
LCD_SPL_SPR pulse width is fixed and aligned to the first data of the line.

Figure 22-13. Horizontal Timing

MCF5329 Reference Manual, Rev 3

22.3.12 LCDC PWM Contrast Control Register (LCD_PCCR)

The PWM contrast control register is used to control the signal output at the contrast pin, which controls the contrast of the LCD panel.

Address: 0xFC0A_C02C (LCD_PCCR)   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	CLS_HI_WIDTH	LD MSK	0	0	0	0	SCR	CC_ EN	PW
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
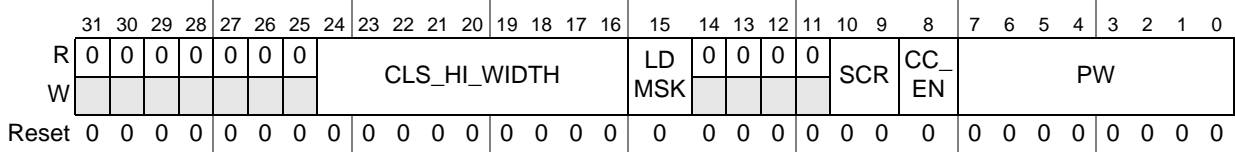
Figure 22-14. LCD PWM Contrast Control Register (LCD_PCCR)

Table 22-15. LCD_PCCR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24–16 CLS_HI_ WIDTH	LCD_CLS high pulse width. Controls the pulse width of LCD_CLS in units of LCD_LSCLK. The actual pulse width is CLS_HI_WDITH + 1.
15 LDMSK	LD mask. Enables/disables the line data (LCD_D[17:0]) output to zero for the Sharp TFT panel power-off sequence. 0 LCD_D[17:0] is normal. 1 LCD_D[17:0] always equals 0.
14–11	Reserved, must be cleared.
10–9 SCR	Source select. Selects the input clock source for the PWM counter. The PWM output frequency is equal to the frequency of the input clock divided by 256. 00 Line pulse 01 Pixel clock 10 LCD clock ($f_{sys/3}$) 11 Reserved
8 CC_EN	Contrast control enable. Enables/disables the contrast control function. 0 Contrast control is off. 1 Contrast control is on.
7–0 PW	Pulse width. Controls the pulse width of the built-in pulse width modulator, which controls the contrast of the LCD screen.

22.3.13 LCDC DMA Control Register (LCD_DCR)

There is a 32×32 bit line buffer in the LCDC that stores DMA data from system memory. The DMA control register controls the DMA burst length and when to trigger a DMA burst in terms of the number of data bytes left in the pixel buffer.

Address: 0xFC0A_C030 (LCD_DCR)   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	BURST	0	0	0	0	0	0	0	0	0	0			HM			0	0	0	0	0	0	0	0	0	0	0			TM		
W																																
Reset	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	0	1	0	0

Figure 22-15. LCD DMA Control Register (LCD_DCR)

Table 22-16. LCD_DCR Field Descriptions

Field	Description
31 BURST	Burst length. Determines whether the burst length is fixed or dynamic. 0 Burst length is dynamic 1 Burst length is fixed
30–21	Reserved, must be cleared.
20–16 HM	DMA high mark. Establishes the high mark for DMA requests. For dynamic burst length, after the DMA request is made, data is loaded and the pixel buffer continues to be filled until the number of empty longwords left in the DMA FIFO is equal to the high mark minus 2. The minimum HM setting in dynamic burst is 3. For fixed burst length, the burst length (in longwords) of each request is equal to the HM setting and its value must be larger than TM.
15–5	Reserved, must be cleared.
4–0 TM	DMA trigger mark. Sets the low-level mark in the pixel buffer to trigger a DMA request. The low-level mark equals the number of longwords left in the pixel buffer.

NOTE

For SDRAM access, a fixed burst length of 16 is recommended:

BURST = 1; HM = 16; TM = 4

For a heavily loaded bus that requires SDRAM access, a dynamic burst length is recommended:

BURST = 0; HM = 4; TM = 8

22.3.14 LCDC Refresh Mode Control Register (LCD_RMCR)

The refresh mode control register is used to enable/disable self-refresh mode.

Address: 0xFC0A_C034 (LCD_RMCR)   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	SELF_ REF
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. LCD Refresh Mode Control Register (LCD_RMCR)

Table 22-17. LCD_RMCR Field Descriptions

Field	Description
31–1	Reserved, must be cleared.
0 SELF_REF	Self-refresh mode enable. 0 Disable self-refresh. 1 Enable self-refresh.

NOTE

1. On entering self-refresh mode, the LCD_LSCLK and LCD_D[17:0] signals stay low. HYSN and VSYN operate normally.

2. Except for the SSA, BGLUT, and GWLUT registers, all configurations must be performed before enabling the LCDC to avoid a malfunction.

3. The SSA must always match the address range of the RAM selected. If the user wants to switch between various types of RAM, the LCDC must be disabled before switching.

22.3.15 LCDC Interrupt Configuration Register (LCD_ICR)

The interrupt configuration register is used to configure the interrupt conditions.

Address: 0xFC0A_C038 (LCD_ICR)　　　　　　　　　　　　　　　　　　　　Access: User read/write
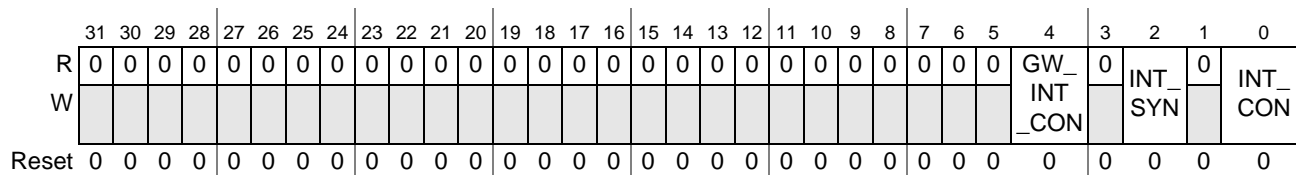
Figure 22-17. LCD Interrupt Configuration Register (LCD_ICR)

Table 22-18. LCD_ICR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 GW_INT_ CON	Graphic window interrupt condition. Determines if an interrupt condition is set at the beginning or the end of graphic window condition. 0 Interrupt flag is set when the end of a graphic window is reached 1 Interrupt flag is set when the beginning of a graphic window is reached
3	Reserved, must be cleared.

Table 22-18. LCD_ICR Field Descriptions (continued)

Field	Description
2 INT_SYN	Interrupt source. Determines if an interrupt flag is set during last data/first data of frame loading or on last data/first data of frame output to the LCD panel. Please refer to the below table for INTSYN/INTCON setting usage. There is a latency between loading the last/first data of frame to output to LCD panel. 0 Interrupt flag is set on loading the last data/first data of frame from memory 1 Interrupt flag is set on output of the last data/first data of frame to LCD panel | INTSYN	INTCON	Description | |---|---|---| | 0 | 0 | Interrupt flag is set on loading last data of frame from memory. | | 0 | 1 | Interrupt flag is set on loading first data of frame from memory. | | 1 | 0 | Interrupt flag is set on output of last data of frame to LCD panel. | | 1 | 1 | Interrupt flag is set on output of first data of frame to LCD panel. |
1	Reserved, must be cleared.
0 INT_CON	Interrupt condition. Determines if an interrupt condition is set at the beginning or the end of frame condition. Refer to table in the INT_SYN field description for INTSYN/INTCON setting usage. 0 Interrupt flag is set when the End of Frame (EOF) is reached 1 Interrupt flag is set when the Beginning of Frame (BOF) is reached

22.3.16 LCDC Interrupt Enable Register (LCD_IER)

The LCDC interrupt enable Register is used to enable the LCDC interrupt signal generated to the interrupt controller. When the interrupt is masked, the LCDC does not generate the interrupt request, but its status can be observed in the interrupt status register.

Address: 0xFC0A_C03C (LCD_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	0	0	0	0	0	0	0	0	GW UDR	GW ERR	GW EOF	GW BOF	UDR	ERR	EOF	BOF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-18. LCD Interrupt Endable Register (LCD_IER)

Table 22-19. LCD_IER Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 GWUDR	Graphic window underrun error interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
6 GWERR	Graphic window error response interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
5 GWEOF	Graphic window end-of-frame interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
4 GWBOF	Graphic window beginning-of-frame interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
3 UDR	Underrun error interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
2 ERR	Error response interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
1 EOF	End-of-frame interrupt enable. 0 Mask interrupt. 1 Enable interrupt.
0 BOF	Beginning-of-frame interrupt enable. 0 Mask interrupt. 1 Enable interrupt.

22.3.17   LCDC Interrupt Status Register (LCD_ISR)

The read-only interrupt status register indicates whether an interrupt has occurred. The status bit is set when the interrupt condition is met. If any bit in this register is set and the corresponding bit in the LCD_IER register is set, an LCD interrupt is asserted to the interrupt controller. The status bit is cleared by reading the register.

Address: 0xFC0A_C040 (LCD_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	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	GW UDR	GW ERR	GW EOF	GW BOF	UDR	ERR	EOF	BOF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-19. LCD Interrupt Status Register (LCD_ISR)

Table 22-20. LCD_ISR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 GWUDR	Graphic window underrun error. Indicates whether the LCDC FIFO in the graphic window plane has hit an underrun condition. This is when the data output rate is faster than the data input rate to the FIFO of the graphic window plane. Underrun can cause erroneous data output to LCD_D. The LCD_D data output rate must be adjusted to prevent this error. 0 Interrupt has not occurred. 1 Interrupt has occurred.
6 GWERR	Graphic window error response interrupt. Indicates whether the LCDC has issued a read data request in graphic window and has received a bus error. It is cleared by reading the status register, at power on reset, or when the LCDC is disabled. 0 Interrupt has not occurred. 1 Interrupt has occurred.
5 GWEOF	Graphic window end-of-frame interrupt. Indicates whether the end of graphic window has been reached. It is cleared by reading the status register, at power on reset, or when the LCDC is disabled. 0 Interrupt has not occurred. 1 Interrupt has occurred.
4 GWBOF	Graphic window beginning-of-frame interrupt. Indicates whether the beginning of the graphic window has been reached. It is cleared by reading the status register, at power on reset, or when the LCDC is disabled. 0 Interrupt has not occurred. 1 Interrupt has occurred.
3 UDR	Underrun error interrupt. Indicates whether the LCDC FIFO has hit an underrun condition. This is when the data output rate is faster than the data input rate to the FIFO. Underrun can cause erroneous data output to LCD_D. The LCD_D data output rate must be adjusted to prevent this error. 0 Interrupt has not occurred. 1 Interrupt has occurred.
2 ERR	Error response interrupt. Indicates whether the LCDC has issued a read data request and has received a bus error. It is cleared by reading the status register, at power on reset, or when the LCDC is disabled. 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 22-20. LCD_ISR Field Descriptions (continued)

Field	Description
1 EOF	End-of-frame interrupt. Indicates whether the end of frame has been reached. It is cleared by reading the status register, at power on reset, or when the LCDC is disabled. 0 Interrupt has not occurred. 1 Interrupt has occurred.
0 BOF	Beginning-of-frame interrupt. Indicates whether the beginning of frame has been reached. It is cleared by reading the status register, at power on reset, or when the LCDC is disabled. 0 Interrupt has not occurred. 1 Interrupt has occurred.

22.3.18 LCDC Graphic Window Start Address Register (LCD_GWSAR)

The LCDC graphic window start address register defines the starting address of the graphic window image.

Address: 0xFC0A_C050 (LCD_GWSAR)   Access: User read/write
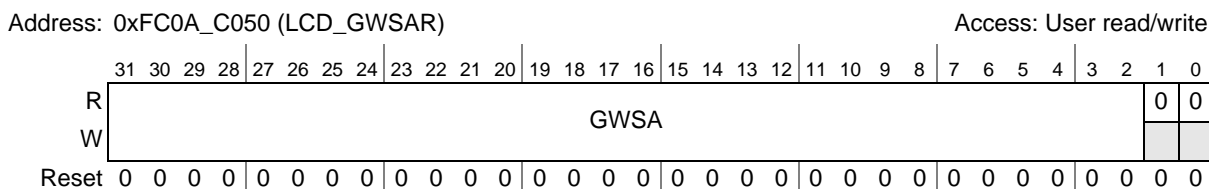
Figure 22-20. LCD Graphic Window Start Address Register (LCD_GWSAR)

Table 22-21. LCD_GWSAR Field Descriptions

Field	Description
31–2 GWSA	Graphic window start address on LCD screen. Holds the starting address of the graphic window picture. This field must start at a location that enables a complete graphic window picture to be stored in a 4 Mbyte memory boundary (A[21:0]). A[31:22] has a fixed value for the graphic window picture's image.
1–0	Reserved, must be cleared.

22.3.19 LCDC Graphic Window Size Register (LCD_GWSR)

The LCDC graphic window size register defines the height and width of the graphic window on the LCD screen.

Address: 0xFC0A_C054 (LCD_GWSR)   Access: User read/write
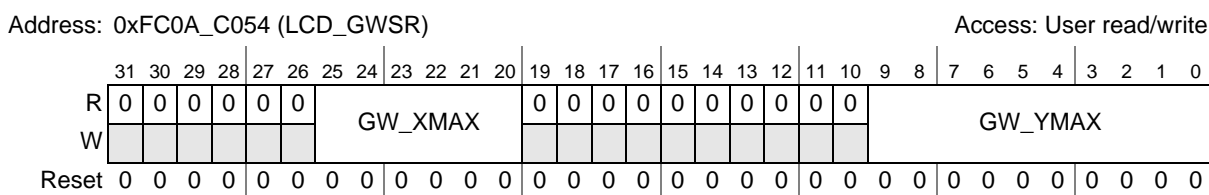
Figure 22-21. LCD Graphic Window Size Register (LCD_GWSR)

Table 22-22. LCD_GWSR Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25–20 GW_XMAX	Graphic window width divided by 16. Holds graphic window x-axis size, divided by 16. For black-and-white panels (1 bpp), GW_XMAX[0] is ignored, forcing the x-axis of the screen size to be a multiple of 32 pixels/line. The graphic window size cannot be set to 0. **Note:** The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x32.
19–10	Reserved, must be cleared.
9–0 GW_YMAX	Graphic window height. Specifies the height of the graphic window in terms of pixels or lines. The lines are numbered from 1 to GW_YMAX for a total of GW_YMAX lines. The graphic window size cannot be set to 0. **Note:** The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x258.

22.3.20 LCDC Graphic Window Virtual Page Width Register (LCD_GWVPW)

The graphic window virtual page width register defines the width of the virtual page for the graphic window picture on the LCD screen.

Address: 0xFC0A_C058 (LCD_GWVPW)   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						GWVPW				
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 22-22. LCD Graphic Window Virtual Page Width Register (LCD_GWVPW)

Table 22-23. LCD_GWVPW Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9–0 GWVPW	Graphic window virtual page width. Defines the virtual page width of the graphic window picture. The VPW bits represent the number of 32-bit longwords required to hold the data for one virtual line. GWVPW is used in calculating the starting address representing the beginning of each line of the graphic window picture.

22.3.21 LCDC Graphic Window Panning Offset Register (LCD_GWPOR)

The panning offset register sets up the panning for the image.

Address: 0xFC0A_C05C (LCD_GWPOR)   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		GWPO			
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 22-23. LCD Graphic Window Panning Offset Register (LCD_GWPOR)

Table 22-24. LCD_GWPOR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4–0 GWPO	Graphic window panning offset. Defines the number of bits that the graphic window data from memory is panned to the left before processing. GWPO is read by the LCDC once at the beginning of each frame. For example, in 4 bpp mode, setting GWPO = 16 shifts 16 bits, which means panning the image by 4 pixels left. **Note:** Use the LGWSAR register to shift the data more than 32 bits or for 18 bpp panning. To achieve panning of the final image by N bits: <table><tr><th>Bits Per Pixel</th><th>GWPO</th><th>Effective # of pixels Panned on Image</th></tr><tr><td>1</td><td>N</td><td>N</td></tr><tr><td>2</td><td>2N</td><td>N</td></tr><tr><td>4</td><td>4N</td><td>N</td></tr><tr><td>8</td><td>8N</td><td>N</td></tr><tr><td>12/16</td><td>16N</td><td>N</td></tr></table>

22.3.22 LCDC Graphic Window Position Register (LCD_GWPR)

The LCDC graphic window position register is used to determine the starting position of the graphic window on the LCD panel.

Address: 0xFC0A_C060 (LCD_GWPR)   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					GWXP						0	0	0	0	0	0					GWYP					
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
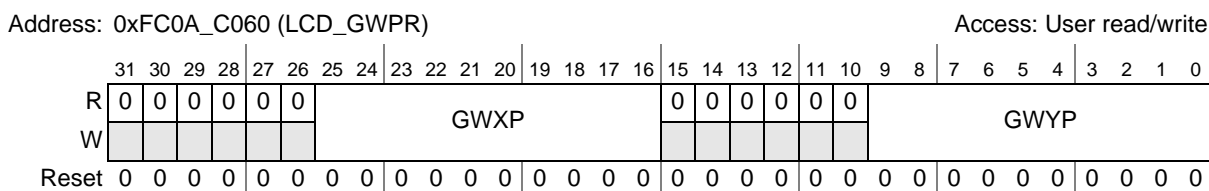
Figure 22-24. LCD Graphic Window Position Register (LCD_GWPR)

Table 22-25. LCD_GWPR Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25–16 GWXP	Graphic window X-position. Represents the graphic window's horizontal starting position in pixel count (from 0 to XMAX).
15–10	Reserved, must be cleared.
9–0 GWYP	Graphic window Y-position. Represents the graphic window's vertical starting position in lines (from 0 to YMAX).

22.3.23 LCDC Graphic Window Control Register (LCD_GWCR)

The LCD graphic window control register defines various aspects of the graphic window.

NOTE

The graphic window can only be enabled while the clock to the LCD controller is disabled (MISCCR[LCDCHEN] is cleared). See Chapter 9, "Chip Configuration Module (CCM)," for details on this bit. The graphics window can be disabled at any time.

Address: 0xFC0A_C064 (LCD_GWCR)   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	GWAV		GW CKE	GWE	GW_RVS	0	0	0	GWCKR	GWCKG	GWCKB
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
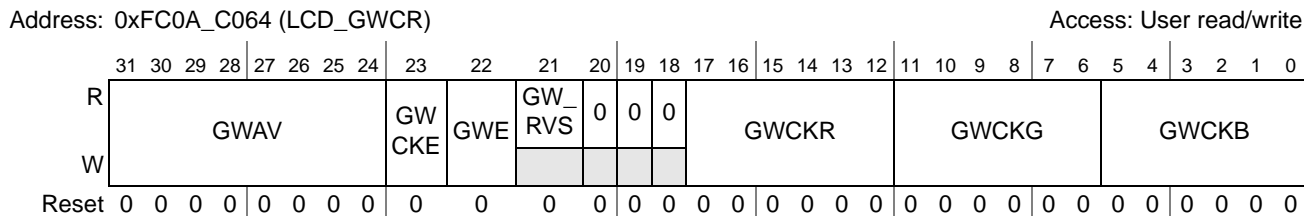
Figure 22-25. LCD Graphic Window Control Register (LCD_GWCR)

Table 22-26. LCD_GWCR Field Descriptions

Field	Description
31–24 GWAV	Graphic window alpha value. Defines the alpha value of graphic window used for alpha blending between the graphic window and the background plane 0 Graphic window totally transparent, not displayed on the LCD screen. 1 Graphic window totally opaque, completely visible on the LCD screen.
23 GWCKE	Graphic window color keying enable. Enable or disable graphic window color keying. 0 Disable color keying of graphic window. 1 Enable color keying of graphic window.
22 GWE	Graphic window enable. Enable or disable graphic window displayed on screen. 0 Disable graphic window on screen. 1 Enable graphic window on screen.
21 GW_RVS	Graphic window reverse vertical scan. Selects the graphic window vertical scan direction as normal or reverse (graphic window image flips along the x-axis). The LGWSAR must be changed accordingly. 0 Vertical scan in normal direction. 1 Vertical scan in reverse direction.
20–18	Reserved, must be cleared.
17–12 GWCKR	Graphic window color keying red component. Defines the red component of graphic window color keying. 0x00 No red … 0x3F Full red
11–6 GWCKG	Graphic window color keying green component. Defines the green component of graphic window color keying. 0x00 No green … 0x3F Full green
5–0 GWCKB	Graphic window color keying blue component. Defines the blue component of graphic window color keying. 0x00 No blue … 0x3F Full blue

22.3.24 LCDC Graphic Window DMA Control Register (LCD_GWDCR)

There is a 32×32 bit line buffer in the LCDC that stores graphic window data from system memory. The graphic window DMA control register controls the DMA burst length and when to trigger a DMA burst in terms of the number of data bytes left in the pixel buffer.

Address: 0xFC0A_C068 (LCD_GWDCR)   Access: User read/write
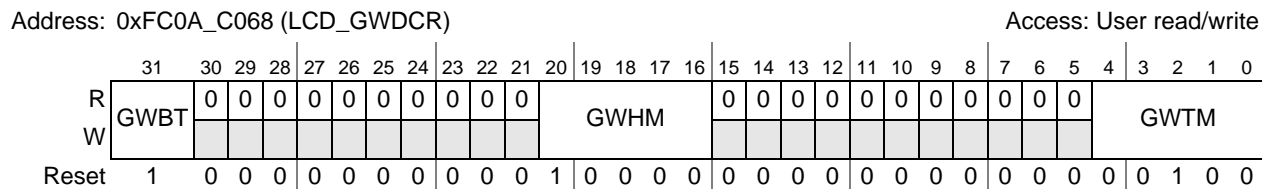
Figure 22-26. LCD Graphic Window DMA Control Register (LCD_GWDCR)

Table 22-27. LCD_GWDCR Field Descriptions

Field	Description
31 GWBT	Graphic window DMA burst type. Determines whether the burst length is fixed or dynamic in graphic window plane. 0 Burst length is dynamic 1 Burst length is fixed
30–21	Reserved, must be cleared.
20–16 GWHM	Graphic window DMA high mark. Establishes the high mark for DMA requests. For dynamic burst length, after the DMA request is made, data is loaded and the graphic window FIFO continues to be filled until the number of empty longwords left in the graphic window FIFO is equal to the high mark minus 2. The minimum GWHM setting in dynamic burst is 3. For fixed burst length, the burst length (in longwords) of each request is equal to the GWHM setting and its value must be larger than GWTM.
15–5	Reserved, must be cleared.
4–0 GWTM	Graphic window DMA trigger mark. Sets the low level mark in the graphic window FIFO to trigger a DMA request. The low level mark equals the number of longwords left in the pixel buffer.

22.3.25 Mapping RAM Registers (BGLUT and GWLUT)

There are two separate mapping RAMs in the LCD controller, the background lookup table (BGLUT) and the graphic window lookup table (GWLUT). The BGLUT is for the background plane and the mapping table is addressable from 0xFC0A_C800–0xFC0A_CBFC. The GWLUT is for the graphic window and its mapping table is addressable from 0xFC0A_CC00–0xFC0A_CFFC.

These mapping RAMs are used for mapping 4-bit codes for grayscale to 16 gray shades, and for mapping 4-bit color and 8-bit color to 16 colors and 256 colors, respectively, out of a palette of 4096 (passive panels) or 256K (active panels).

The mapping RAM contains 256 entries and each entry is accessed with longword transactions only and the address must be longword aligned. Unimplemented bits are read as 0. All read and write data use the least significant 12 or 18 bits.

NOTE

Byte or word access to the RAM corrupts its contents.

In 4 bpp mode, the first 16 RAM entries are used. In 8 bpp mode, all 256 RAM entries are used. The color RAM is not initialized at reset. Only the following settings use the mapping RAMs:

- 4 bpp gray-scale mode
- 4 bpp passive matrix color mode
- 8 bpp passive matrix color mode
- 4 bpp active matrix color mode
- 8 bpp active matrix color mode

22.3.25.1 Four Bits Per Pixel Gray-Scale Mode

In four bits per pixel gray-scale mode, a 4-bit code represents a gray-scale level. The first 16 mapping RAM entries must be written to define the codes for all 16 combinations.

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																													GPM			
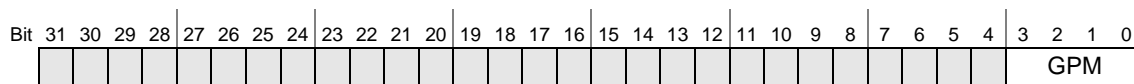
Table 22-28. Four Bits Per Pixel Gray-Scale Mode

Field	Description
31–4	Reserved, must be cleared.
3–0 GPM	Gray palette map. Represents the gray-scale level for a given pixel code.

22.3.25.2 Four Bits Per Pixel Passive Matrix Color Mode

In four bits per pixel passive matrix color mode, a 4-bit code represents a 12-bit color. Because only four bits are used to encode the color, a maximum of 16 colors can be selected out of a palette of 4096. The first 16 mapping RAM entries must be written to define the codes for the 16 available combinations.

Bit	31	30	29	28	27	26	25	24	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				G				B			
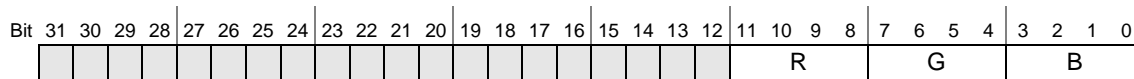
Table 22-29. Four Bits Per Pixel Passive Matrix Color Mode

Field	Description
31–12	Reserved, must be cleared.
11–8 R	Red level (color display). Represents the red component level in the color.
7–4 G	Green level (color display). Represents the green component level in the color.
3–0 B	Blue level (color display). Represents the blue component level in the color.

22.3.25.3 Eight Bits Per Pixel Passive Matrix Color Mode

In eight bits per pixel passive matrix color mode, an 8-bit code represents a 12-bit color. Because eight bits are used to encode the color, a maximum of 256 colors can be selected out of a palette of 4096. All 256 mapping RAM entries must be written to define the codes for the 256 available combinations.

Bit	31	30	29	28	27	26	25	24	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				G				B			
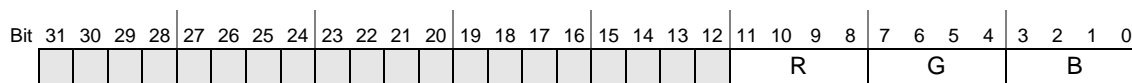
Table 22-30. Eight Bits Per Pixel Passive Matrix Color Mode

Field	Description
31–12	Reserved, must be cleared.
11–8 R	Red level (color display). Represents the red component level in the color.
7–4 G	Green level (color display). Represents the green component level in the color.
3–0 B	Blue level (color display). Represents the blue component level in the color.

22.3.25.4 Four Bits Per Pixel Active Matrix Color Mode

In four bits per pixel active color mode, a 4-bit code represents a 18-bit color. Because only four bits are used to encode the color, a maximum of 16 colors can be selected out of a palette of 256K. The first 16 mapping RAM entries must be written to define the codes for the 16 available combinations.

Bit	31	30	29	28	27	26	25	24	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						G						B					
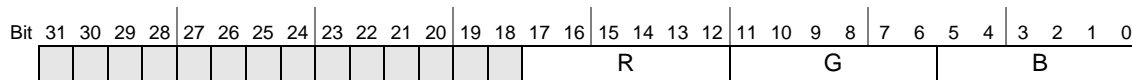
Table 22-31. Four Bits Per Pixel Active Matrix Color Mode

Field	Description
31–18	Reserved, must be cleared.
17–12 R	Red level (color display). Represents the red component level in the color.
11–6 G	Green level (color display). Represents the green component level in the color.
5–0 B	Blue level (color display). Represents the blue component level in the color.

22.3.25.5 Eight Bits Per Pixel Active Matrix Color Mode

In eight bits per pixel active color mode, an 8-bit code represents an 18-bit color. Because eight bits are used to encode the color, a maximum of 256 colors can be selected out of a palette of 256K. All 256 mapping RAM entries must be written to define the codes for the 256 available combinations.

Bit	31	30	29	28	27	26	25	24	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						G					B				
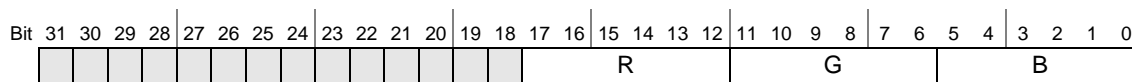
Table 22-32. Eight Bits Per Pixel Active Matrix Color Mode

Field	Description
31–18	Reserved, must be cleared.
17–12 R	Red level (color display). Represents the red component level in the color.
11–6 G	Green level (color display). Represents the green component level in the color.
5–0 B	Blue level (color display). Represents the blue component level in the color.

22.4 Functional Description

The following sections describe the operation of the LCD controller with various industry standard LCD displays.

22.4.1 LCD Screen Format

The number of pixels forming the screen width and screen height of the LCD panel are software programmable. Figure 22-27 shows the relationship between the screen size and memory window.
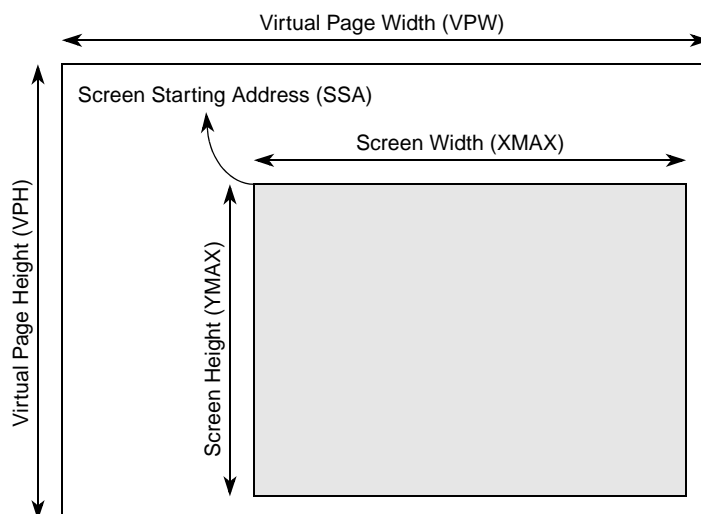
Figure 22-27. LCD Screen Format

The screen width (XMAX) and screen height (YMAX) parameters specify the LCD panel size. The LCDC begins scanning the display memory at the location pointed to by the screen starting address register (LCD_SSAR), represented by the shaded area in Figure 22-27, for display on the LCD panel.

The maximum page width is specified by the virtual page width (VPW) parameter. Virtual page height (VPH) does not affect the LCDC and is limited only by memory size. By changing the SSA register, a

screen-sized window can be vertically or horizontally scrolled (panned) anywhere inside the virtual page boundaries. The software must control the starting address in the SSA properly so that the scanning logic's system memory pointer (SMP) stays within the VPW and VPH limits to prevent the display of strange artifacts on the screen.

VPH is used by the programmer only for boundary checks. There is no VPH parameter internal to the LCDC.

VPW is used in calculating the RAM starting address representing the beginning of each displayed line. SSA sets the address of data for the first line of a frame. For each subsequent line, VPW is added to an accumulation initialized by the SSA to yield the starting address of that line.

22.4.2 Graphic Window on Screen

Graphic window is supported in LCD color panel screens for viewfinder and graphic hardware cursor functions. Similar to the screen, the virtual page width, graphic window start address and graphic window width and height are software programmable. The position of the graphic window on screen is specified by the graphic window position register. Figure 22-28 shows how the graphic window is configured and placed on the screen.
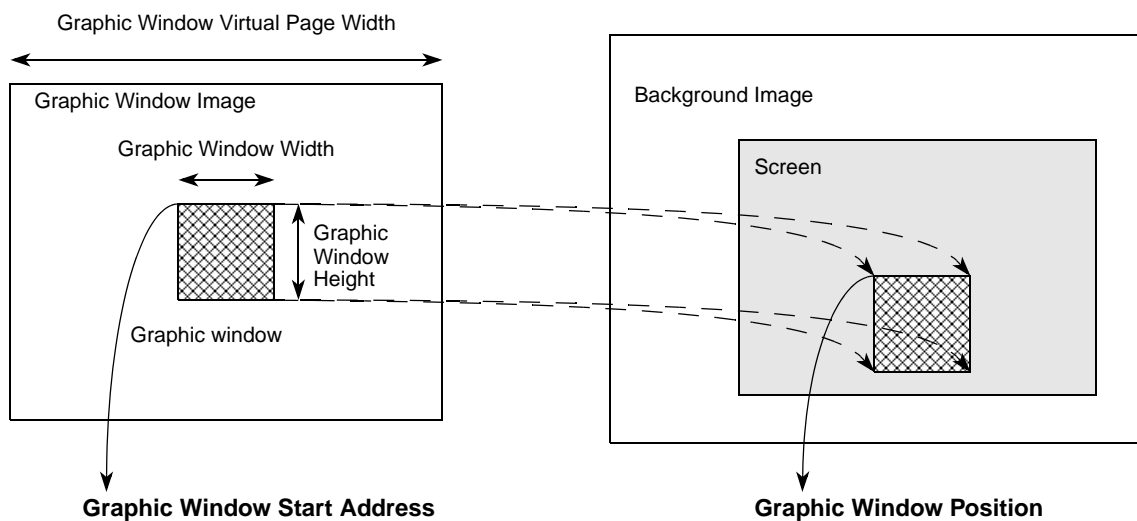
Figure 22-28. Graphic Window on Screen

The graphic window and background plane can be alpha blended. The alpha value is programmable on a window basis, which means all the pixels in the graphic window have the same level of transparency. There are a total of 256 levels of transparency to be configured. In addition, one of the pixel colors can be chosen for color keying in which the selected pixel color in the graphics window is made totally transparent. One of the applications can be a graphical hardware cursor.

NOTE

The graphic window and background images must have the same bpp setting.

22.4.3 Panning

Panning offset (POS) is expressed in bits, not pixels, so when operating in any mode other than 1 bpp, only even pixel boundaries are valid. In 12 bpp mode, the pixels are aligned to 16-bit boundaries, and POS also must align to these boundaries.

SSA and POS are located in isolated registers and are double buffered because they are dynamic parameters likely to change while the LCDC is running. New values of SSA and POS do not take effect until the beginning of the next frame. A typical panning algorithm includes an interrupt at the beginning of the frame. In the interrupt service routine, POS and/or SSA are updated (the old values are internally latched). The updates take effect on the next frame.

22.4.4 Display Data Mapping

The LCDC supports 1/2/4 bpp in monochrome mode and 4/8/12/16/18 bpp in color mode. System memory data mapping in 2/4/8/12/16/18 bpp modes is shown in Figure 22-29.

NOTE

In 12 bpp mode, 16 bits of memory are used for each set of 12 bits, to leave 4 bits unused. In 18 bpp mode, 32 bits of memory are used for each pixel, leaving 14 bits unused. Refer to Figure 22-30 and Figure 22-31.

Figure 22-29. Pixel Location on Display Screen

1 bpp Mode

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

2 bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	P0		P1		P2		P3	
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	P4		P5		P6		P7	
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	P8		P9		P10		P11	
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
	P12		P13		P14		P15	

4 bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	P0				P1			
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	P2				P3			
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	P4				P5			
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
	P6				P7			

8 bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	P0							
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	P1							
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	P2							
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
	P3							

12 bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	Green0 [3]	Green0 [2]	Green0 [1]	Green0 [0]	Blue0 [3]	Blue0 [2]	Blue0 [1]	Blue0 [0]
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
					Red0 [3]	Red0 [2]	Red0 [1]	Red0 [0]
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	Green1 [3]	Green1 [2]	Green1 [1]	Green1[0]	Blue1 [3]	Blue1 [2]	Blue1 [1]	Blue1 [0]
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
					Red1 [3]	Red1 [2]	Red1 [1]	Red1 [0]

16 bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	Green0 [2]	Green0 [1]	Green0 [0]	Blue0 [4]	Blue0 [3]	Blue0 [2]	Blue0 [1]	Blue0 [0]
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	Red0 [4]	Red0 [3]	Red0 [2]	Red0 [1]	Red0 [0]	Green0 [5]	Green0 [4]	Green0 [3]
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	Green1 [2]	Green1 [1]	Green1 [0]	Blue1 [4]	Blue1 [3]	Blue1 [2]	Blue1 [1]	Blue1 [0]
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24
	Red1 [4]	Red1 [3]	Red1 [2]	Red1 [1]	Red1 [0]	Green1 [5]	Green1 [4]	Green1 [3]

Figure 22-30. Display Data Mapping 1 bpp Through 16 bpp Modes

18 bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	Blue [5]	Blue [4]	Blue [3]	Blue [2]	Blue [1]	Blue [0]		
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	Green [5]	Green [4]	Green [3]	Green [2]	Green [1]	Green [0]		
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	Red [5]	Red [4]	Red [3]	Red [2]	Red [1]	Red [0]		
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24

Microsoft PAL_BGR 18bpp Mode

Byte Address	Sample Bit-to-Pixel Mapping							
0	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	Red [5]	Red [4]	Red [3]	Red [2]	Red [1]	Red [0]		
1	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
	Green [5]	Green [4]	Green [3]	Green [2]	Green [1]	Green [0]		
2	Bit 23	Bit 22	Bit 21	Bit 20	Bit 19	Bit 18	Bit 17	Bit 16
	Blue [5]	Blue [4]	Blue [3]	Blue [2]	Blue [1]	Blue [0]		
3	Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	Bit 26	Bit 25	Bit 24

Figure 22-31. Display Data Mapping for 18 bpp Modes

22.4.5 Black-and-White Operation

The 1 bpp mode is also known as black-and-white mode because each pixel is always fully on or fully off.

22.4.6 Gray-Scale Operation

The LCDC generates a maximum of 16 gray levels. These gray levels are defined by 2 or 4 bits of display data for each pixel. Using 2 bpp, the LCDC displays 4 shades of gray, and using 4 bpp, the LCDC displays all 16 shades. The shades of gray are obtained by controlling the number of frames in which the pixel is on over a period of 16 frames. This method is known as frame rate control (FRC). For more information on FRC, see Section 22.4.8, "Frame Rate Modulation Control (FRC)

The use of the mapping RAM is shown in Figure 22-32. When using 2 bpp, the 2-bit code is mapped to one of four gray levels, and when using 4 bpp, the 4-bit code is mapped to one of 16 gray levels. Because crystal formulations and driving voltages vary, the visual gray effect may or may not be linearly related to the frame rate. A logarithmic scale such as 0, 1/4, 1/2 and 1 might be more pleasing than a linearly spaced scale such as 0, 5/16, 11/16 and 1 for certain graphics.

Figure 22-32 illustrates gray-scale pixel generation. The flexible mapping scheme allows the user to optimize the visual effect for a specific panel or application.
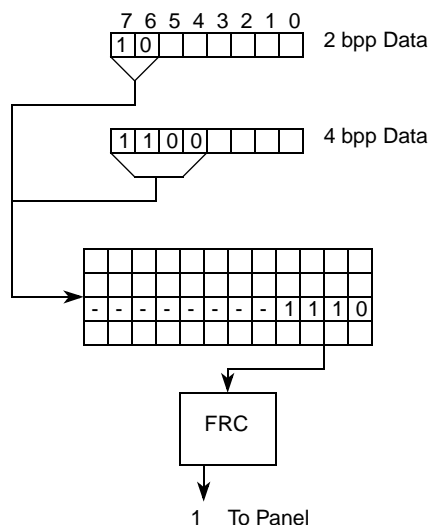
Figure 22-32. Gray-Scale Pixel Generation

22.4.7 Color Generation

The value corresponding to each color pixel on the screen is represented by a 4-, 8-, 12-, 16- or 18-bit code in the display memory.

For 4- and 8-bit modes the LCDC's color mapping RAM is used to map the data to a 12-bit and 18-bit RGB code for passive and active matrix color displays respectively. For 4-bit and 8-bit passive matrix color displays, the 12-bit RGB code from the mapping RAM is output to the FRC blocks that independently process the code corresponding to the red, green, and blue components of each pixel to generate the required shade and intensity.

For 4-bit and 8-bit active matrix display, the 18-bit output from the mapping RAM is output to the panel.

For 12-bit mode for passive matrix color display, the mapping RAM is bypassed and output directly to the FRC block.

For 12-, 16- and 18-bit active matrix color display, pixel data is simply moved from display memory to the LCDC output bus.

Figure 22-33 and Figure 22-34 illustrate passive matrix and active matrix color pixel generation.
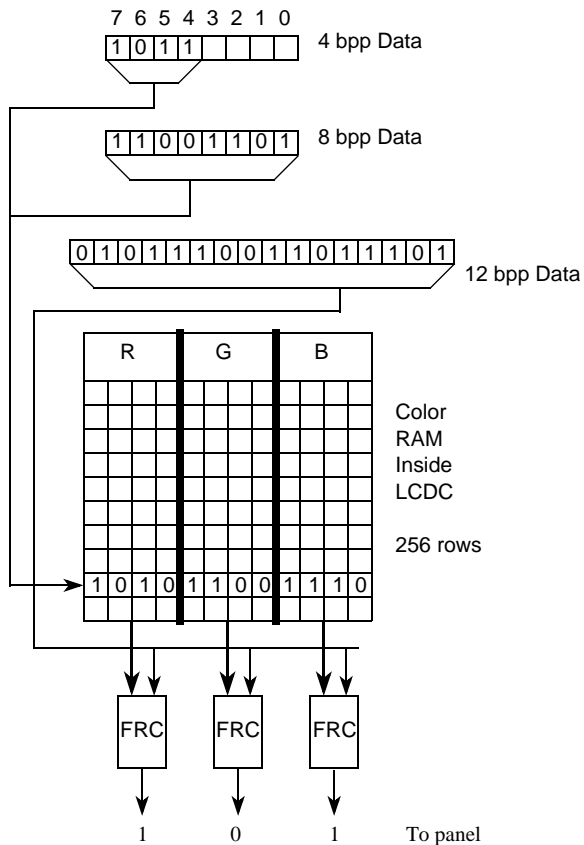
Figure 22-33. Passive Matrix Color Pixel Generation
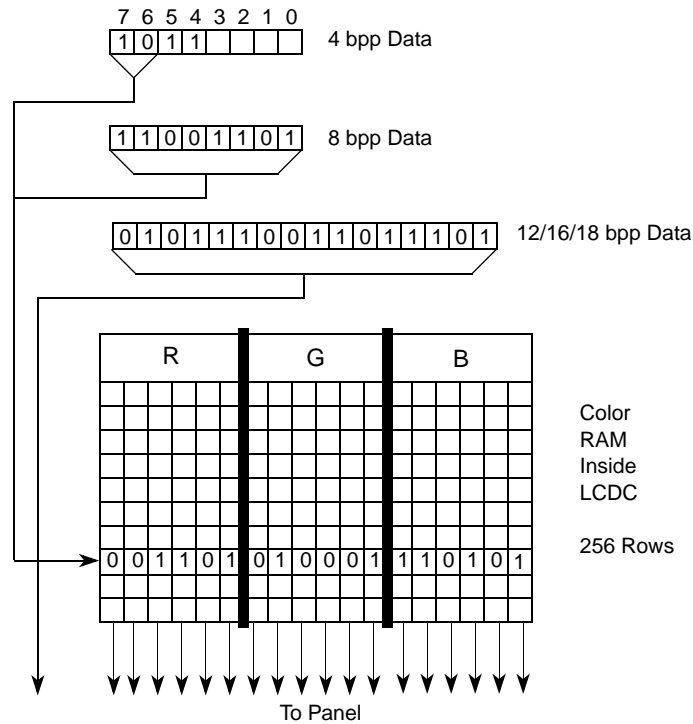
Figure 22-34. Active Matrix Color Pixel Generation

22.4.8 Frame Rate Modulation Control (FRC)

Circuitry inside the LCDC generates intermediate gray-scale colors on the panel by adjusting the density of zeroes and ones that appear over the frames. The LCDC can generate 16 simultaneous gray-scale levels.

Table 22-33. Gray Palette Density

Gray Code (Hexadecimal)	Density	Density (Decimal)
0	0	0
1	1/8	0.125
2	1/5	0.2
3	1/4	0.25
4	1/3	$0.\overline{333}$
5	2/5	0.4
6	4/9	$0.\overline{444}$
7	1/2	0.5
8	5/9	$0.\overline{555}$
9	3/5	0.6
A	2/3	$0.\overline{666}$
B	3/4	0.75

Table 22-33. Gray Palette Density (continued)

Gray Code (Hexadecimal)	Density	Density (Decimal)
C	4/5	0.8
D	7/8	0.875
E	14/15	$0.9\overline{33}$
F	1	1
Note: Overbars indicate repeating decimal numbers.		

22.4.9 Panel Interface Signals and Timing

The LCDC continuously provides pixel data to the LCD panel via the LCD panel interface. Panel interface signals are illustrated in .
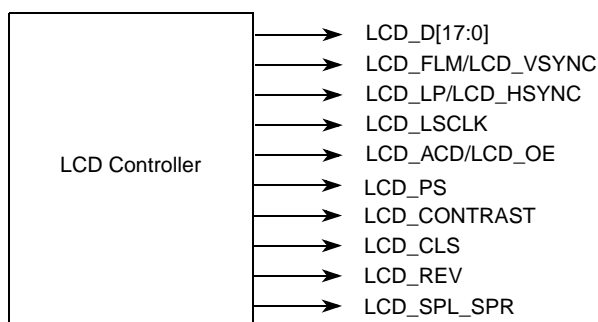
Figure 22-35. LCDC Interface Signals

The format, timing, and polarity of the panel interface signals are programmable. There are two basic modes, passive and active, selected by the TFT register bit. The user must also select grayscale mode or color mode.

The LCD_SPL_SPR, LCD_PS, LCD_CLS, and LCD_REV interface signals are dedicated for Sharp HR-TFT 240 × 320 panels only.

22.4.9.1 Pin Configuration for LCDC

Figure 22-35 shows the signals used for the LCDC. These pins are multiplexed with other functions on the device, and must be configured for LCDC operation before they can be used.

22.4.9.2 Passive Matrix Panel Interface Signals

Figure 22-36 shows the LCD interface timing for monochrome panels and Figure 22-37 shows the LCD interface timing for passive matrix color panels. Signal polarities are shown positive, however it can be reversed by clearing the bits in the panel configuration register (LCD_PCR). The data bus timing for passive panels is determined by the shift clock (LCD_LSCLK), line pulse (LCD_LP), first line marker (LCD_FLM), alternate crystal direction (LCD_ACD), and line data (LCD_D) signals.

Operation of the panel interface is accomplished in the following steps:

1. LCD_LSCLK clocks the pixel data into the display driver's internal shift register.
2. LCD_LP signifies the end of the current line of serial data and latches the shifted pixel data into a wide latch.
3. LCD_FLM marks the first line of the displayed page. The LCD_D (and the associated LCD_LP), enclosed by the LCD_FLM signal, marks the first line of the current frame.
4. LCD_ACD toggles after a pre-programmed number of LCD_FLM pulses. This signal refreshes the LCD panel.

NOTE

The LCD_D bus width is programmable to 1, 2, 4, or 8 bits in monochrome mode (the COLOR bit in the panel configuration register is set to 0). Data is justified to the least significant bits of the LCD_D[17:0] bus. Passive color displays use a fixed 2-2/3 pixels of data per 8-bit vector as shown in Figure 22-37.
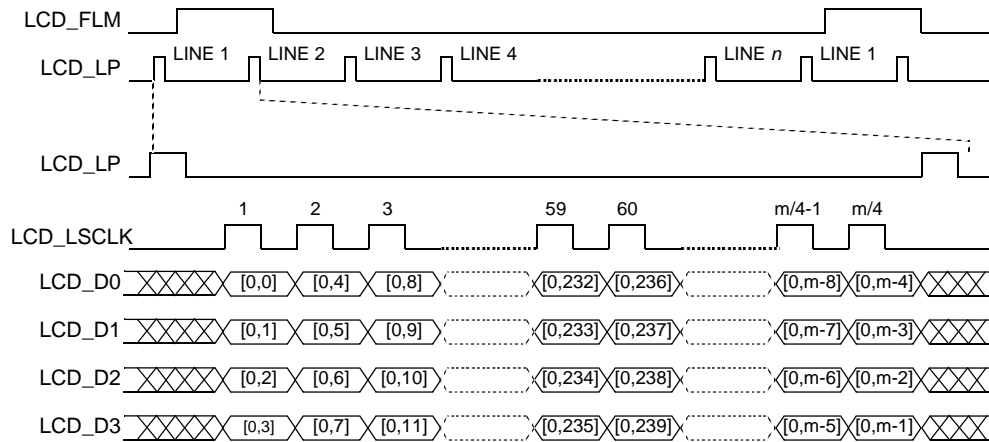
Figure 22-36. LCDC Interface Timing for 4-bit Data Width Gray-Scale Panels
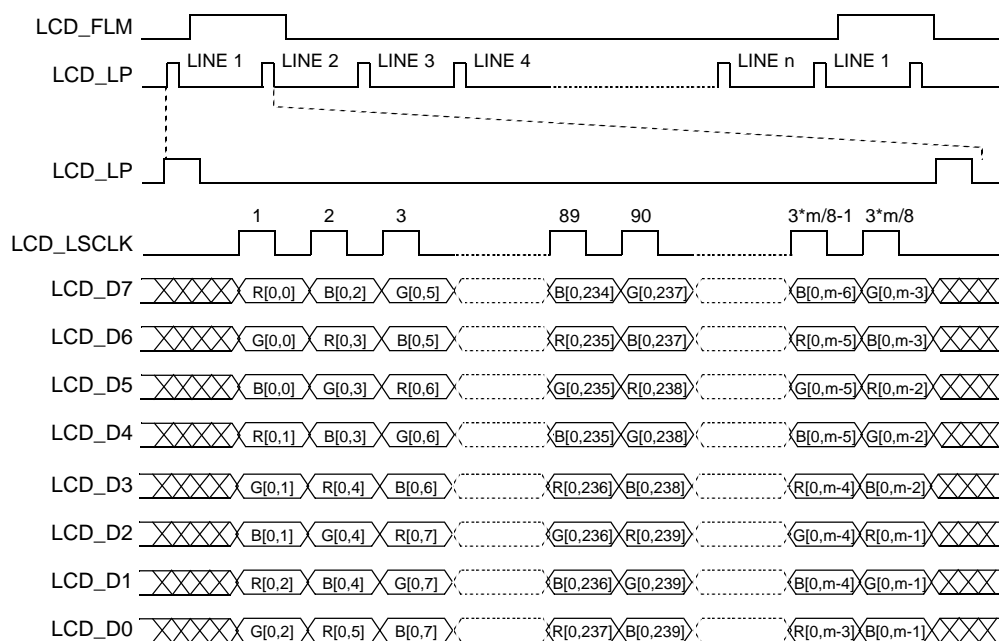
Figure 22-37. LCDC Interface Timing for 8-bit Data Passive Matrix Color Panels

22.4.9.3 Passive Panel Interface Timing

Figure 22-38 shows the horizontal timing (timing of one line), including the line pulse (LCD_LP) and the data. The width of LCD_LP and delays before and after LCD_LP are programmable.

The parameters used for panel interface timing are:

- XMAX (X size) defines the number of pixels per line. XMAX is the total number of pixels per line.
- H_WAIT_1 defines the delay from the end of data output to the beginning of LCD_LP.
- H_WIDTH (horizontal sync pulse width) defines the width of the LCD_FLM pulse, and H_WIDTH must be at least 1.
- H_WAIT_2 defines the delay from the end of LCD_LP to the beginning of data output.

NOTE

All parameters are defined in unit of pixel clock period $\left(\frac{\text{LCD\_PCR[PCD]} + 1}{f_{\text{sys/3}}}\right)$ unless stated otherwise.
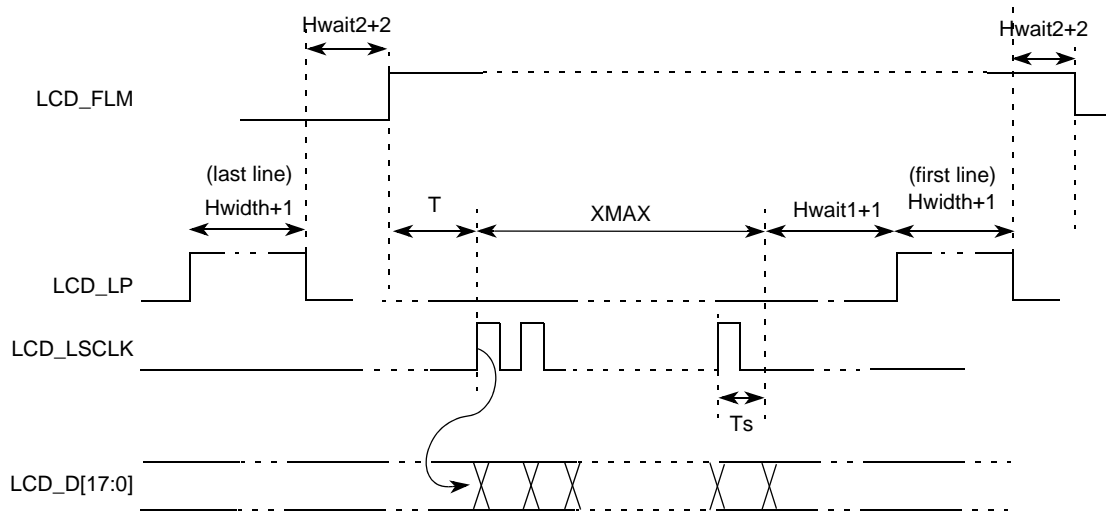
Figure 22-38. Horizontal Sync Pulse Timing in Passive Mode

End of last line

Start of frame
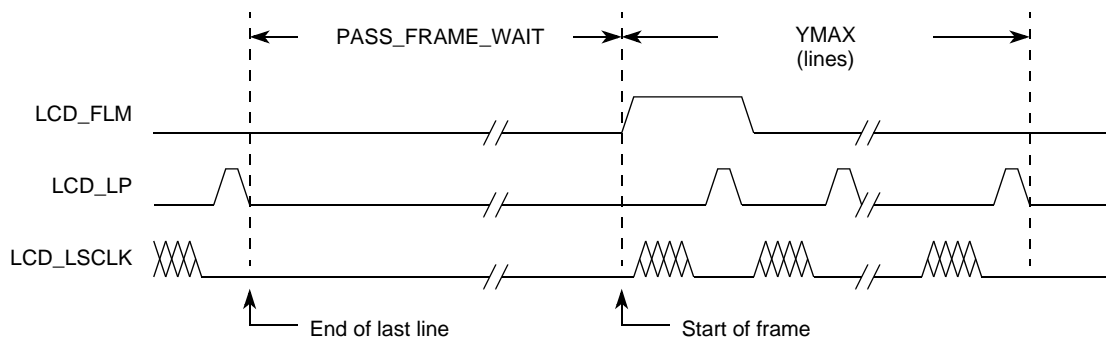
Figure 22-39. Vertical Sync Pulse Timing in Passive Mode

22.4.10 8 bpp Mode Color STN Panel

22.4.10.1 Active Matrix Panel Interface Signals

Figure 22-40 shows the LCD interface timing for an active matrix color TFT panel. In this figure, signals are shown with negative polarity (FLMPOL=1, LPPOL=1, CLKPOL=0, OEPOL=1). In TFT mode, the LCD_LSCLK is automatically inverted. The panel interface timing for active matrix panels is sometimes referred to as a digital CRT and is controlled by the shift clock (LCD_LSCLK), horizontal sync pulse (LCD_HSYNC, the LCD_LP pin in passive mode), vertical sync pulse (LCD_VSYNC, the LCD_FLM pin in passive mode), output enable (LCD_OE, the LCD_ACD pin in passive mode), and line data (LCD_D) signals. The sequence of events for active matrix interface timing is:

1. LCD_LSCLK latches data into the panel on its negative edge (when positive polarity is selected). In active mode, LCD_LSCLK runs continuously.
2. LCD_HSYNC causes the panel to start a new line.

3. LCD_VSYNC causes the panel to start a new frame. It always encompasses at least one LCD_HSYNC pulse.

4. LCD_OE functions as an output enable signal to the CRT. This output enable signal is similar to the blanking output in a CRT and enables the data to be shifted onto the display. When disabled, the data is invalid and the trace is off.

In 4- and 8-bit mode, the LCD_D[17:12] bits define red, the LCD_D[11:6] bits define green, and the LCD_D[5:0] bits define blue. In 12-bit mode, the LCD_D[17:14] bits define red, the LCD_D[11:8] bits define green, and the LCD_D[5:2] bits define blue. In 16-bit mode, the LCD_D[17:13] bits define red, the LCD_D[11:6] bits define green, and the LCD_D[5:1] bits define blue.

The actual TFT color channel assignments are shown in Table 22-34. The unused bits are fixed at 0.

Table 22-34. TFT Color Channel Assignments

	LCD_D[17:0]																	
	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4 bpp	R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0
8 bpp	R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0
12 bpp	R3	R2	R1	R0	–	–	G3	G2	G1	G0	–	–	B3	B2	B1	B0	–	–
16 bpp	R4	R3	R2	R1	R0	–	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0	–
18 bpp	R5	R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B5	B4	B3	B2	B1	B0
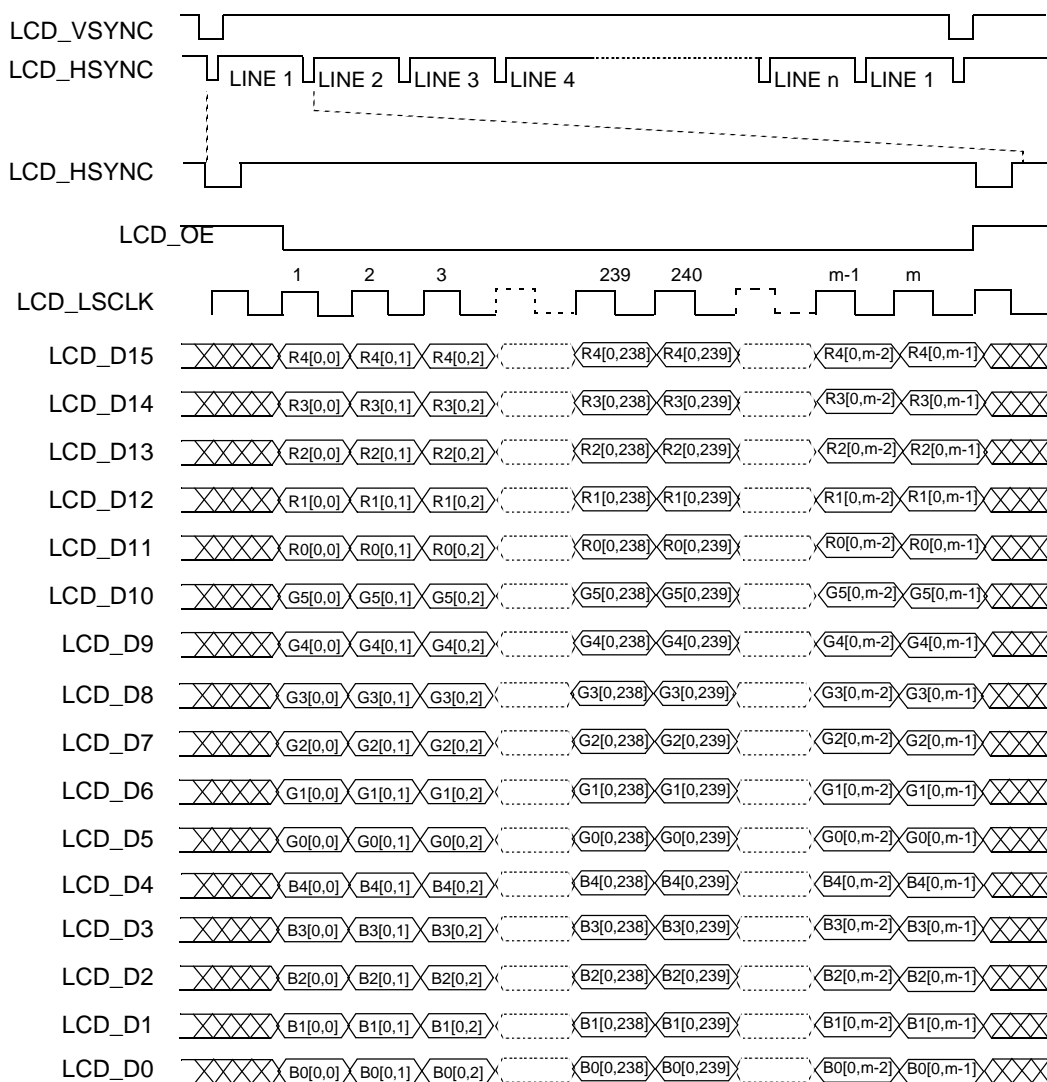
Figure 22-40. LCDC Interface Timing for Active Matrix Color Panels

22.4.10.2 Active Panel Interface Timing

Figure 22-41 shows the horizontal timing (timing of one line), including the horizontal sync pulse and the data. The width of LCD_HSYNC and delays before and after LCD_HSYNC are programmable.

The timing signal parameters are defined as follows:

- H_WIDTH defines the width of the LCD_HSYNC pulse and must be at least 1.
- H_WAIT_2 defines the delay from the end of LCD_HSYNC to the beginning of the LCD_OE pulse.
- H_WAIT_1 defines the delay from end of LCD_OE to the beginning of the LCD_HSYNC pulse.
- XMAX defines the (total) number of pixels per line.

NOTE

All parameters are defined in pixel periods $\left(\frac{\text{LCD\_PCR[PCD]} + 1}{f_{\text{sys}/3}}\right)$, not LCD_LSCLK periods.
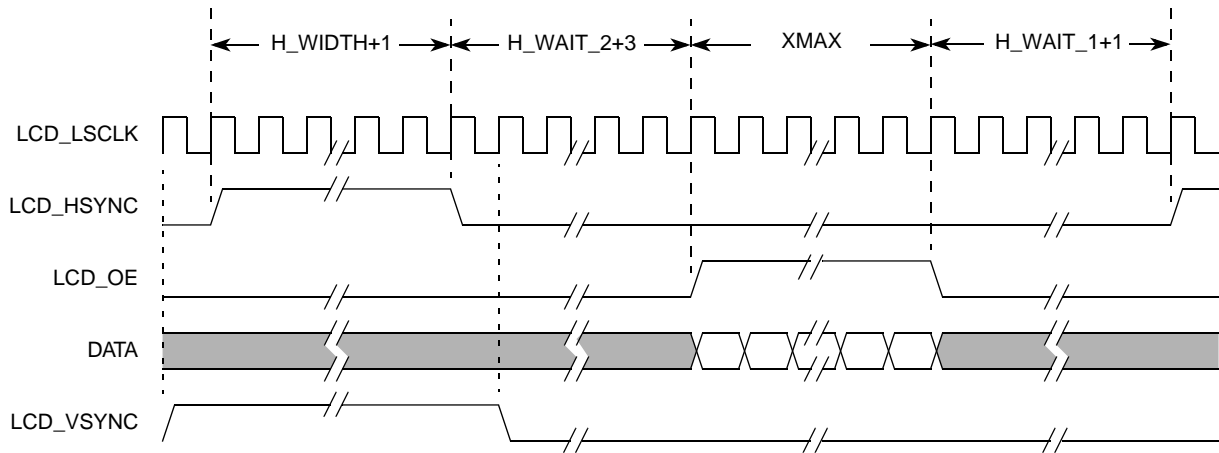
Figure 22-41. Horizontal Sync Pulse Timing in TFT Mode

Figure 22-42 shows the vertical timing (timing of one frame). The delay from the end of one frame until the beginning of the next is programmable. The memory timing signal parameters are:

- V_WAIT_1 is a delay measured in lines. For V_WAIT_1= 1 there is a delay of one LCD_HSYNC (time = one line period) before LCD_VSYNC. The LCD_HSYNC pulse is output during the V_WAIT_1 delay.

- For V_WIDTH (vertical sync pulse width) = 0, LCD_VSYNC encloses one LCD_HSYNC pulse. For V_WIDTH = 2, LCD_VSYNC encloses two LCD_HSYNC pulses.

- V_WAIT_2 is a delay measured in lines. For V_WAIT_2 = 1, there is a delay of one LCD_HSYNC (time = one line period) after LCD_VSYNC. The LCD_HSYNC pulse is output during the V_WAIT_2 delay.
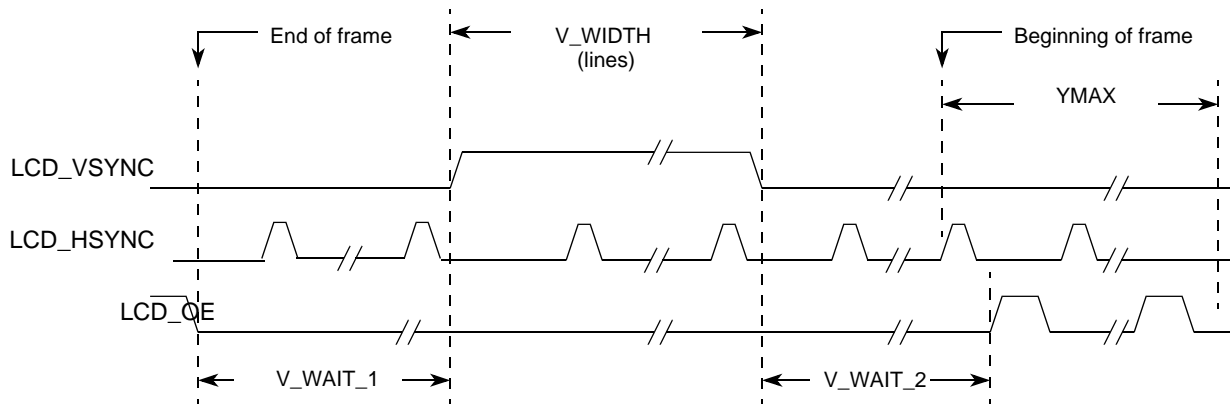
Figure 22-42. Vertical Sync Pulse Timing TFT Mode

Chapter 23
FlexCAN

23.1 Introduction

The FlexCAN is a communication controller implementing the controller area network (CAN) protocol, an asynchronous communications protocol used in automotive and industrial control systems. It is a high speed (1 Mbps), short distance, priority-based protocol that can communicate using a variety of mediums (such as fiber optic cable or an unshielded twisted pair of wires). The FlexCAN supports the standard and extended identifier (ID) message formats specified in the CAN protocol specification, revision 2.0, part B.

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness, and required bandwidth. A general working knowledge of the CAN protocol revision 2.0 is assumed in this document. For details, refer to the CAN protocol revision 2.0 specification.

23.1.1 Block Diagram

A block diagram describing the various submodules of the FlexCAN module is shown in Figure 23-1. Each submodule is described in detail in subsequent sections.
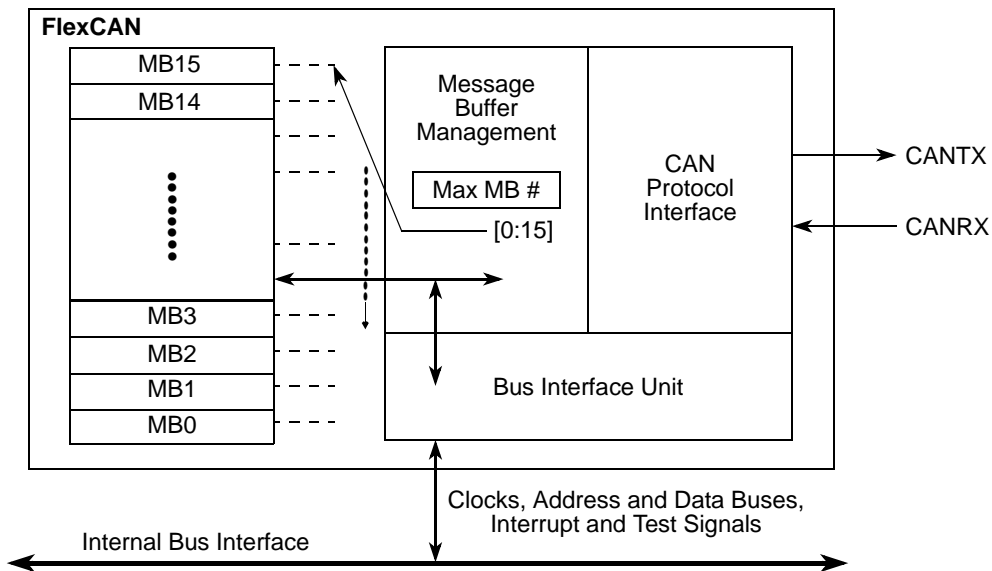
Figure 23-1. FlexCAN Block Diagram

The message buffer architecture is shown in Figure 23-2.
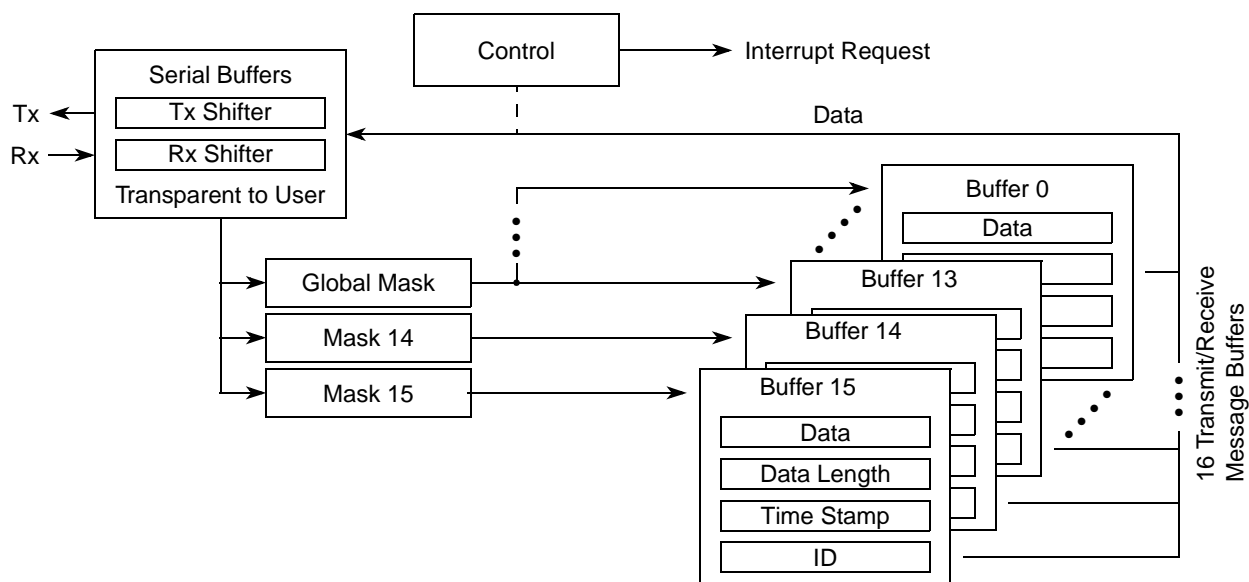
Figure 23-2. FlexCAN Message Buffer Architecture

23.1.1.1 The CAN System

A typical CAN system is shown below in Figure 23-3. Each CAN station is connected physically to the CAN bus through a transceiver. The transceiver provides the transmit drive, waveshaping, and receive/compare functions required for communicating on the CAN bus. It can also provide protection against damage to the FlexCAN caused by a defective CAN bus or defective stations.
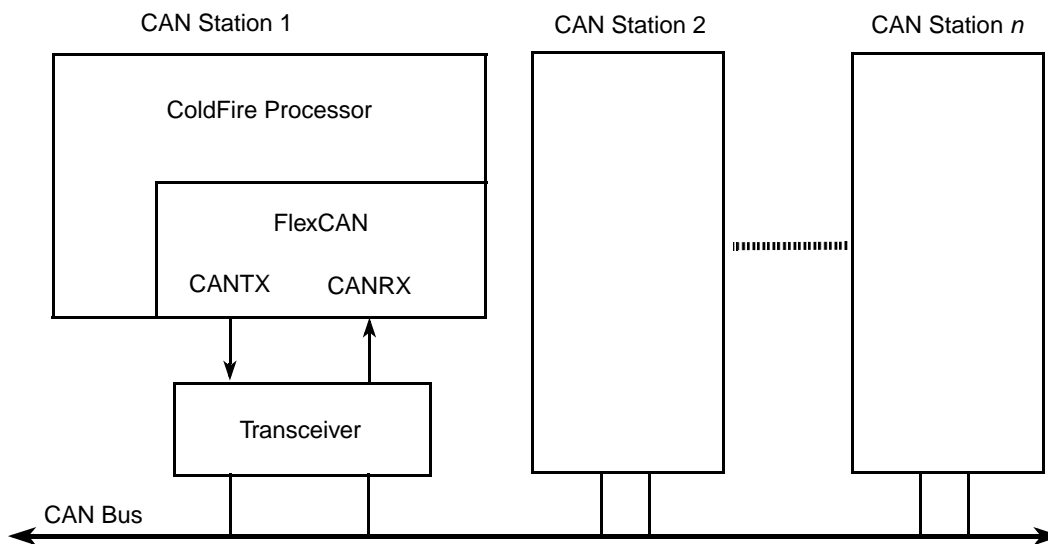
Figure 23-3. Typical CAN System

23.1.2 Features

Following are the main features of the FlexCAN module:

- Full implementation of the CAN protocol specification version 2.0B
    - Standard data and remote frames (up to 109 bits long)
    - Extended data and remote frames (up to 127 bits long)
    - 0–8 bytes data length
    - Programmable bit rate up to 1 Mbps
    - Content-related addressing
- Up to 16 flexible message buffers of zero to eight bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Listen-only mode capability
- Three programmable mask registers: global (for MBs 0–13), special for MB14, and special for MB15
- Programmable transmission priority scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit, free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Open network architecture
- Multimaster bus
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages

23.1.3 Modes of Operation

23.1.3.1 Normal Mode

In normal mode, the module operates receiving and/or transmitting message frames, errors are managed normally, and all the CAN protocol functions are enabled. User and supervisor modes differ in the access to some restricted control registers.

23.1.3.2 Freeze Mode

Freeze mode is entered by setting:

- CANMCR[FRZ], and
- CANMCR[HALT], or by asserting the $\overline{\text{BKPT}}$ signal.

After entry into freeze mode is requested, the FlexCAN waits until an intermission or idle condition exists on the CAN bus, or until the FlexCAN enters the error passive or bus off state. After one of these conditions exists, the FlexCAN waits for the completion of all internal activity such as arbitration, matching, move-in, and move-out. When this happens, the following events occur:

- The FlexCAN stops transmitting/receiving frames.
- The prescaler is disabled, thus halting all CAN bus communication.
- The FlexCAN ignores its Rx pins and drives its Tx pins as recessive.
- The FlexCAN loses synchronization with the CAN bus and the NOTRDY and FRZACK bits in CANMCR are set.
- The CPU is allowed to read and write the error counter registers (in other modes they are read-only).

After engaging one of the mechanisms to place the FlexCAN in freeze mode, the user must wait for the FRZACK bit to be set before accessing any other registers in the FlexCAN; otherwise, unpredictable operation may occur. In freeze mode, all memory mapped registers are accessible.

To exit freeze mode, the $\overline{\text{BKPT}}$ line must be negated or the HALT bit in CANMCR must be cleared. After freeze mode is exited, the FlexCAN resynchronizes with the CAN bus by waiting for 11 consecutive recessive bits before beginning to participate in CAN bus communication.

23.1.3.3 Module Disabled Mode

This mode disables the FlexCAN module; it is entered by setting CANMCR[MDIS]. If the module is disabled during freeze mode, it shuts down the system clocks, sets the LPMACK bit, and clears the FRZACK bit.

If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in idle or bus-off state, or else waits for the third bit of intermission and then checks it to be recessive
- Waits for all internal activities such as arbitration, matching, move-in, and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the system clocks

The bus interface unit continues to operate, enabling the CPU to access memory-mapped registers, except the free-running timer, the error counter register, and the message buffers, which cannot be accessed when the module is disabled. Exiting from this mode is done by negating the MDIS bit, which resumes the clocks and negate the LPMACK bit.

23.1.3.4 Loop-back Mode

The module enters this mode when the LPB bit in the control register is set. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Transmit and receive interrupts are generated.

23.1.3.5 Listen-only Mode

In listen-only mode, transmission is disabled, all error counters are frozen and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station are received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. Because the module does not influence the CAN bus in this mode, the device is capable of functioning like a monitor or for automatic bit-rate detection.

23.2 External Signal Description

Each FlexCAN module has two I/O signals connected to the external MPU pins: CANTX and CANRX. CANTX transmits serial data to the CAN bus transceiver, while CANRX receives serial data from the CAN bus transceiver.

23.3 Memory Map/Register Definition

The FlexCAN module address space is split into 128 bytes starting at the base address, and 256 bytes starting at the base address + 0x80. Out of the lower 128 bytes, only part is occupied by various registers. The upper 256 bytes are fully used for the message buffer structures, as described in Section 23.3.9, "Message Buffer Structure."

Table 23-1. FlexCAN Memory Map

Address FlexCAN	Register	Width (bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
Supervisor-only Access Registers
0xFC02_0000	FlexCAN Module Configuration Register (CANMCR)	32	Y	Y	R/W	0xD890_000F	23.3.1/23-6
Supervisor/User Access Registers
0xFC02_0004	FlexCAN Control Register (CANCTRL)	32	Y	N	R/W	0x0000_0000	23.3.2/23-8
0xFC02_0008	Free Running Timer (TIMER)	32	Y	Y	R/W	0x0000_0000	23.3.3/23-10
0xFC02_0010	Rx Global Mask (RXGMASK)	32	Y	N	R/W	0x1FFF_FFFF	23.3.4/23-11
0xFC02_0014	Rx Buffer 14 Mask (RX14MASk)	32	Y	N	R/W	0x1FFF_FFFF	23.3.4/23-11
0xFC02_0018	Rx Buffer 15 Mask (RX15MASK)	32	Y	N	R/W	0x1FFF_FFFF	23.3.4/23-11
0xFC02_001C	Error Counter Register (ERRCNT)	32	Y	Y	R/W	0x0000_0000	23.3.6/23-13
0xFC02_0020	Error and Status Register (ERRSTAT)	32	Y	Y	R/W	0x0000_0000	23.3.6/23-13
0xFC02_0028	Interrupt Mask Register (IMASK)	32	Y	Y	R/W	0x0000_0000	23.3.7/23-15
0xFC02_0030	Interrupt Flag Register (IFLAG)	32	Y	Y	R/W	0x0000_0000	23.3.8/23-16
0xFC02_0080	Message Buffers 0–15 (MB0–15)	2048	N	N	R/W	—	23.3.9/23-16

NOTE

The FlexCAN has no hard-wired protection against invalid bit/field programming within its registers. Specifically, no protection is provided if the programming does not meet CAN protocol requirements.

Programming the FlexCAN control registers is typically done during system initialization, prior to the FlexCAN becoming synchronized with the CAN bus. The configuration registers can be changed after synchronization by halting the FlexCAN module. This is done when the user sets the CANMCR[HALT] bit. The FlexCAN responds by setting the CANMCR[NOTRDY] bit.

23.3.1 FlexCAN Configuration Register (CANMCR)

CANMCR defines global system configurations, such as the module operation mode and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in freeze mode.

Address: 0xFC02_0000 (CANMCR)   Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MDIS	FRZ	0	HALT	NOT RDY	0	SOFT RST	FRZ ACK	SUPV	0	0	LPM ACK	0	0	0	0
W																
Reset	1	1	0	1	1	0	0	0	1	0	0	1	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				
W													MAXMB			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 23-4. FlexCAN Configuration Register (CANMCR)

Table 23-2. CANMCR Field Descriptions

Field	Description
31 MDIS	Module disable. This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the FlexCAN clocks that drive the CAN interface and Message Buffer sub-module. This is the only bit in CANMCR not affected by soft reset. See Section 23.1.3.3, "Module Disabled Mode," for more information. 0 Enable the FlexCAN module, clocks enabled 1 Disable the FlexCAN module, clocks disabled
30 FRZ	Freeze mode enable. When set, the FlexCAN can enter freeze mode when the $\overline{BKPT}$ line is asserted or the HALT bit is set. Clearing this bit causes the FlexCAN to exit freeze mode. Refer to Section 23.1.3.2, "Freeze Mode," for more information. 0 FlexCAN ignores the $\overline{BKPT}$ signal and the CANMCR[HALT] bit. 1 FlexCAN module enabled to enter debug mode.
29	Reserved, must be cleared.

Table 23-2. CANMCR Field Descriptions (continued)

Field	Description
28 HALT	Halt FlexCAN. Setting this bit puts the FlexCAN module into freeze mode. It has the same effect as assertion of the $\overline{BKPT}$ signal. This bit is set after reset and should be cleared after initializing the message buffers and control registers. FlexCAN message buffer receive and transmit functions are inactive until this bit is cleared. While in freeze mode, the CPU has write access to the error counter register (ERRCNT) that is otherwise read-only. 0 The FlexCAN operates normally 1 FlexCAN enters freeze mode if FRZ equals 1
27 NOTRDY	FlexCAN not ready. This bit indicates that the FlexCAN is in disable or freeze mode. This bit is read-only and it is cleared after the FlexCAN exits these modes. 0 FlexCAN is in normal mode, listen-only mode, or loop-back mode. h1 FlexCAN is in disable or freeze mode.
26	Reserved, must be cleared.
25 SOFTRST	Soft reset. When set, the FlexCAN resets its internal state machines (sequencer, error counters, error flags, and timer) and the host interface registers (CANMCR [except the MDIS bit], TIMER, ERRCNT, ERRSTAT, IMASK, and IFLAG). The configuration registers that control the interface with the CAN bus are not changed (CANCTRL, RXGMASK, RX14MASK, RX15MASK). Message buffers are also not changed. This allows SOFTRST to be used as a debug feature while the system is running. Because soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFTRST bit remains set while reset is pending and is automatically cleared when reset completes. The user should poll this bit to know when the soft reset has completed. 0 Soft reset cycle completed 1 Soft reset cycle initiated
24 FRZACK	Freeze acknowledge. Indicates that the FlexCAN module has entered freeze mode. The user should poll this bit after freeze mode has been requested, to know when the module has actually entered freeze mode. When freeze mode is exited, this bit is cleared after the FlexCAN prescaler is enabled. This is a read-only bit. 0 The FlexCAN has exited freeze mode and the prescaler is enabled. 1 The FlexCAN has entered freeze mode, and the prescaler is disabled.
23 SUPV	Supervisor/user data space. Places the FlexCAN registers in supervisor or user data space. 0 Registers with access controlled by the SUPV bit are accessible in user or supervisor privilege mode. 1 Registers with access controlled by the SUPV bit are restricted to supervisor mode.
22–21	Reserved, must be cleared.
20 LPMACK	Low power mode acknowledge. Indicates that FlexCAN is disabled. Disabled mode cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPMACK bit to know when the FlexCAN has actually entered low power mode. See Section 23.1.3.3, "Module Disabled Mode," and Chapter 8, "Power Management," for more information. This bit is read-only. 0 FlexCAN not disabled. 1 FlexCAN is in disabled mode.
19–4	Reserved, must be cleared.
3–0 MAXMB	Maximum number of message buffers. Defines the maximum number of message buffers that take part in the matching and arbitration process. The reset value (0xF) is equivalent to16 message buffer (MB) configuration. This field should be changed only while the module is in freeze mode. **Note:** Maximum MBs in Use = MAXMB + 1

23.3.2 FlexCAN Control Register (CANCTRL)

CANCTRL is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, loop back mode, listen-only mode, bus off recovery behavior, and interrupt enabling. It also determines the division factor for the clock prescaler. Most of the fields in this register should only be changed while the module is disabled or in freeze mode. Exceptions are the BOFFMSK, ERRMSK, and BOFFREC bits, which can be accessed at any time.

Address: 0xFC02_0004 (CANCTRL)   Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R				PRESDIV					RJW		PSEG1			PSEG2		
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	BOFF MSK	ERR MSK	CLK_ SRC	LPB	0	0	0	0	SMP	BOFF REC	TSYN	LBUF	LOM	PROPSEG		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
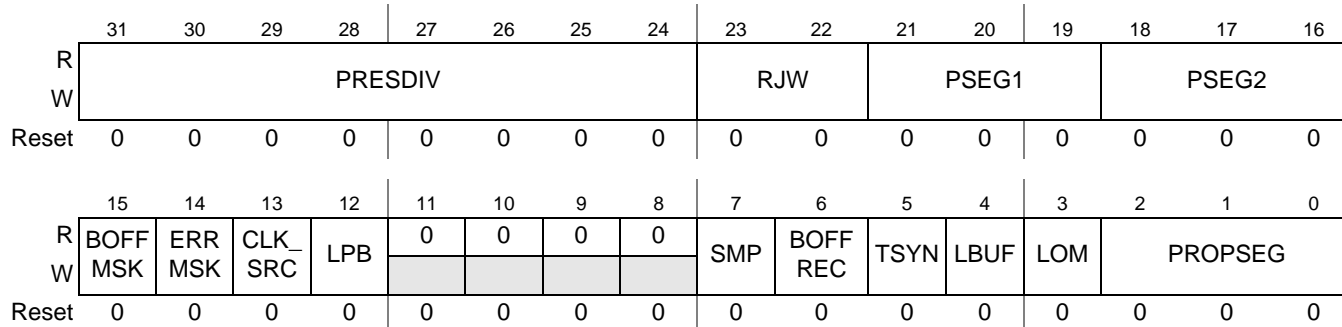
Figure 23-5. FlexCAN Control Register (CANCTRL)

Table 23-3. CANCTRL Field Descriptions

Field	Description
31–24 PRESDIV	Prescaler division factor. Defines the ratio between the clock source frequency (set by CLK_SRC bit) and the serial clock (S clock) frequency. The S clock period defines the time quantum of the CAN protocol. For the reset value, the S clock frequency is equal to the clock source frequency. The maximum value of this register is 0xFF, that gives a minimum S clock frequency equal to the clock source frequency divided by 256. For more information refer to Section 23.3.18, "Bit Timing." $$\text{S clock frequency} = \frac{f_{sys/3} \text{ or EXTAL}}{\text{PRESDIV} + 1} \qquad \textbf{\textit{Eqn. 23-1}}$$
23–22 RJW	Resynchronization jump width. Defines the maximum number of time quanta (one time quantum is equal to the S clock period) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. $$\text{Resync jump width} = (\text{RJW} + 1) \text{ time quanta} \qquad \textbf{\textit{Eqn. 23-2}}$$
21–19 PSEG1	Phase buffer segment 1. Defines the length of phase buffer segment 1 in the bit time. The valid programmable values are 0–7. $$\text{Phase buffer segment 1} = (\text{PSEG1} + 1) \text{ time quanta} \qquad \textbf{\textit{Eqn. 23-3}}$$
18–16 PSEG2	Phase buffer segment 2. Defines the length of phase buffer segment 2 in the bit time. The valid programmable values are 1–7. $$\text{Phase buffer segment 2} = (\text{PSEG2} + 1) \text{ time quanta} \qquad \textbf{\textit{Eqn. 23-4}}$$
15 BOFFMSK	Bus off interrupt mask. 0 Bus off interrupt disabled 1 Bus off interrupt enabled

Table 23-3. CANCTRL Field Descriptions (continued)

Field	Description
14 ERRMSK	Error interrupt mask. 0 Error interrupt disabled 1 Error interrupt enabled
13 CLK_SRC	Clock source. Selects the clock source for the CAN interface to be fed to the prescalar. This bit should only be changed while the module is disabled. 0 Clock source is EXTAL 1 Clock source is the internal bus clock, $f_{sys/3}$
12 LPB	Loop back. Configures FlexCAN to operate in loop-back mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Transmit and receive interrupts are generated. 0 Loop back disabled 1 Loop back enabled
11–8	Reserved, must be cleared.
7 SMP	Sampling mode. Determines whether the FlexCAN module samples each received bit one time or three times to determine its value. 0 One sample, taken at the end of phase buffer segment 1, is used to determine the value of the received bit. 1 Three samples are used to determine the value of the received bit. The samples are taken at the normal sample point and at the two preceding periods of the S-clock; a majority rule is used.
6 BOFFREC	Bus off recovery mode. Defines how FlexCAN recovers from bus off state. If this bit is cleared, automatic recovering from bus off state occurs according to the *CAN Specification 2.0B*. If the bit is set, automatic recovering from bus off is disabled and the module remains in bus off state until the bit is cleared by the user. If the bit is cleared before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFFREC bit had never been set. If the bit is cleared after 128 sequences of 11 recessive bits occurred, FlexCAN re-synchronizes to the bus by waiting for 11 recessive bits before joining the bus. After clearing, the BOFFREC bit can be set again during bus off, but it is only effective the next time the module enters bus off. If BOFFREC was cleared when the module entered bus off, setting it during bus off is not effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0B 1 Automatic recovering from bus off state disabled
5 TSYN	Timer synchronize mode. Enables the mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides the means to synchronize multiple FlexCAN stations with a special SYNC message (global network time). 0 Timer synchronization disabled. 1 Timer synchronization enabled. **Note:** There can be a bit clock skew of four to five counts between different FlexCAN modules that are using this feature on the same network.
4 LBUF	Lowest buffer transmitted first. Defines the ordering mechanism for message buffer transmission. 0 Message buffer with lowest ID is transmitted first 1 Lowest numbered buffer is transmitted first

Table 23-3. CANCTRL Field Descriptions (continued)

Field	Description
3 LOM	Listen-only mode. Configures FlexCAN to operate in listen-only mode. In this mode transmission is disabled, all error counters are frozen, and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station is received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 FlexCAN module is in normal active operation; listen-only mode is deactivated 1 FlexCAN module is in listen-only mode operation
2–0 PROPSEG	Propagation segment. Defines the length of the propagation segment in the bit time. The valid programmable values are 0–7. $$\text{Propagation segment time} = (\text{PROPSEG} + 1) \text{ time-quanta} \qquad \textit{Eqn. 23-5}$$ **Note:** A time-quantum equals 1 S clock period.

23.3.3 FlexCAN Free Running Timer Register (TIMER)

This register represents a 16-bit free running counter that can be read and written to by the CPU. The timer starts from 0x0000 after reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each received or transmitted bit. When there is no message on the bus, it counts using the previously programmed baud rate. During freeze mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register, then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Address: 0xFC02_0008 (TIMER)   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								TIMER								
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
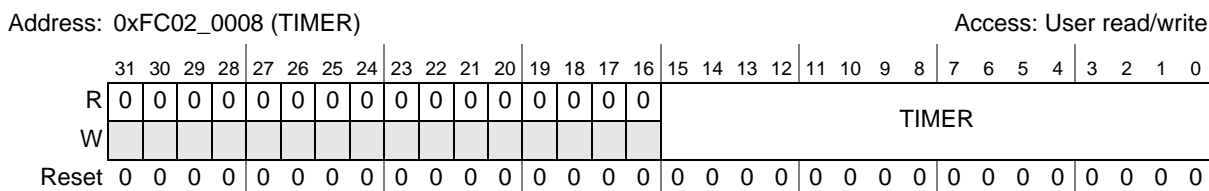
Figure 23-6. FlexCAN Timer Register (TIMER)

Table 23-4. TIMER Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 TIMER	Free running timer. Captured at the beginning of the identifier (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP entry in a message buffer after a successful reception or transmission of a message.

23.3.4 Rx Mask Registers (RXGMASK, RX14MASK, RX15MASK)

These registers are used as acceptance masks for received frame IDs. Three masks are defined: a global mask (RXGMASK) used for Rx buffers 0–13 and two separate masks for buffers 14 (RX14MASK) and 15 (RX15MASK). The meaning of each mask bit is the following:

MIn bit = 0: The corresponding incoming ID bit is don't care.

MIn bit = 1: The corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

These masks are used for standard and extended ID formats. The value of the mask registers should not be changed while in normal operation (only while in freeze mode), as locked frames that matched a message buffer (MB) through a mask may be transferred into the MB (upon release) but may no longer match.

Table 23-5. Mask Examples for Normal/Extended Messages

	Base ID ID28.................ID18	IDE	Extended ID ID17..ID0	Match
MB2-ID	1 1 1 1 1 1 1 1 0 0 0	0		
MB3-ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB4-ID	0 0 0 0 0 0 1 1 1 1 1	0		
MB5-ID	0 0 0 0 0 0 1 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB14-ID	1 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx_Global_Mask	1 1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1	
Rx_Msg in[1]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB3[1]
Rx_Msg in[2]	1 1 1 1 1 1 1 1 0 0 1	0		MB2[2]
Rx_Msg in[3]	1 1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	[3]
Rx_Msg in[4]	0 1 1 1 1 1 1 1 0 0 0	0		[4]
Rx_Msg in[5]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14[5]
RX14MASK	0 1 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0	
Rx_Msg in[6]	1 0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	[6]
Rx_Msg in[7]	0 1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14[7]

[1] Match for Extended Format (MB3).
[2] Match for Normal Format. (MB2).
[3] Mismatch for MB3 because of ID0.
[4] Mismatch for MB2 because of ID28.
[5] Mismatch for MB3 because of ID28, Match for MB14 (Uses RX14MASK).
[6] Mismatch for MB14 because of ID27 (Uses RX14MASK).
[7] Match for MB14 (Uses RX14MASK).

Address: 0xFC02_0010 (RXGMASK)
0xFC02_0014 (RX14MASK)
0xFC02_0018 (RX15MASK)

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		MI Standard ID	MI Extended ID
W						
Reset	0	0	0	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
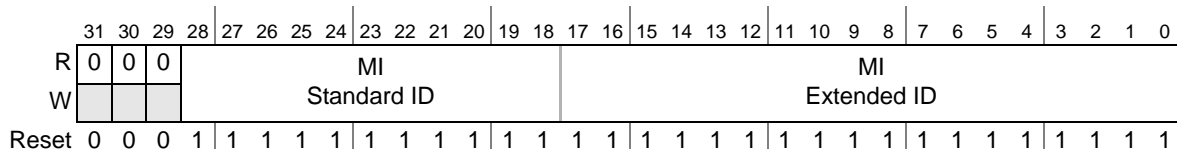
Figure 23-7. FlexCAN Rx Mask Registers (RXGMASK, RX14MASK, RX15MASK)

Table 23-6. RX*xx*MASK Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28–18 MI28–18	Standard ID mask bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–0	Extended ID mask bits. These bits are used to mask comparison only in Extended Format.

23.3.5 FlexCAN Error Counter Register (ERRCNT)

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: transmit error counter (TXECTR) and receive error counter (RXECTR). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read-only, except in freeze mode, where they can be written by the CPU.

Writing to the ERRCNT register while in freeze mode is an indirect operation. The data is first written to an auxiliary register, then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g. transmit error-active or error-passive flag, delay its transmission start time (error-passive), and avoid any influence on the bus when in bus off state. The following are the basic rules for FlexCAN bus state transitions:

- If the value of TXECTR or RXECTR increases to be greater than or equal to 128, the FLTCONF field in the error and status register (ERRSTAT) is updated to reflect error-passive state.
- If the FlexCAN state is error-passive, and TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the ERRSTAT[FLTCONF] field is updated to reflect error-active state.
- If the value of TXECTR increases to be greater than 255, the ERRSTAT[FLTCONF] field is updated to reflect bus off state, and an interrupt may be issued. The value of TXECTR is then reset to zero.
- If FlexCAN is in bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the

ERRSTAT[FLTCONF] field is updated to be error-active, and both error counters are reset to zero. At any instance of a dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECTR value.

- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ERRSTAT[ACKERR] bit). After the transition to error-passive state, the TXECTR does not increment anymore by acknowledge errors. Therefore, the device never goes to the bus off state.

- If the RXECTR increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to error-active state.

Address: 0xFC02_001C (ERRCNT)   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	RXECTR	TXECTR
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
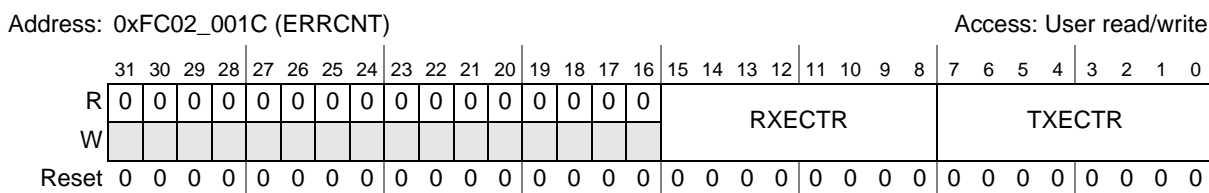
Figure 23-8. FlexCAN Error Counter Register (ERRCNT)

Table 23-7. ERRCNT Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 RXECTR	Receive error counter. Indicates current number of receive errors.
7–0 TXECTR	Transmit error counter. Indicates current number of transmit errors.

23.3.6 FlexCAN Error and Status Register (ERRSTAT)

ERRSTAT reflects various error conditions, some general status of the device, and is the source of three interrupts to the CPU. The reported error conditions (bits 15:10) are those occurred since the last time the CPU read this register. The read action clears bits 15-10. Bits 9–3 are status bits.

Most bits in this register are read only, except for BOFFINT and ERRINT, which are interrupt flags that can be cleared by writing 1 to them. Writing 0 has no effect. Refer to Section 23.4.1, "Interrupts."

Address: 0xFC02_0020 (ERRSTAT)   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	BIT1 ERR	BIT0 ERR	ACK ERR	CRC ERR	FRM ERR	STF ERR	TX WRN	RX WRN	IDLE	TXRX	FLT CONF		0	BOFF INT	ERR INT	0
W														w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-9. FlexCAN Error and Status Register (ERRSTAT)

Table 23-8. ERRSTAT Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15 BIT1ERR	Bit1 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as recessive was received as dominant **Note:** The transmit bit error field is not modified during the arbitration field or the ACK slot bit time of a message, or by a transmitter that detects dominant bits while sending a passive error frame.
14 BIT0ERR	Bit0 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as dominant was received as recessive
13 ACKERR	Acknowledge error. Indicates whether an acknowledgment has been correctly received for a transmitted message. 0 No ACK error was detected since the last read of this register. 1 An ACK error was detected since the last read of this register.
12 CRCERR	Cyclic redundancy check error. Indicates whether or not a CRC error has been detected by the receiver. 0 No CRC error was detected since the last read of this register. 1 A CRC error was detected since the last read of this register.
11 FRMERR	Message form error. Indicates that a form error has been detected by the receiver node, i.e. a fixed-form bit field contains at least one illegal bit. 0 No form error was detected since the last read of this register. 1 A form error was detected since the last read of this register.
10 STFERR	Bit stuff error. 0 No bit stuffing error was detected since the last read of this register. 1 A bit stuffing error was detected since the last read of this register.
9 TXWRN	Transmit error status flag. Reflects the status of the FlexCAN transmit error counter. 0 Transmit error counter < 96 1 TXErrCounter ≥ 96
8 RXWRN	Receiver error status flag. Reflects the status of the FlexCAN receive error counter. 0 Receive error counter < 96 1 RxErrCounter ≥ 96
7 IDLE	Idle status. Indicates when there is activity on the CAN bus. 0 The CAN bus is not idle. 1 The CAN bus is idle.

Table 23-8. ERRSTAT Field Descriptions (continued)

Field	Description
6 TXRX	Transmit/receive status. Indicates when the FlexCAN module is transmitting or receiving a message. TXRX has no meaning when IDLE equals 1. 0 The FlexCAN is receiving a message if IDLE equals 0. 1 The FlexCAN is transmitting a message if IDLE equals 0.
5–4 FLTCONF	Fault confinement state. Indicates the confinement state of the FlexCAN module, as shown below. If the CANCTRL[LOM] bit is set, FLTCONF indicates error-passive. Because the CANCTRL register is not affected by soft reset, the FLTCONF field is not affected by soft reset if the LOM bit is set. 00 Error active 01 Error passive 1x Bus off
3	Reserved, must be cleared.
2 BOFFINT	Bus off interrupt. Used to request an interrupt when the FlexCAN enters the bus off state. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No bus off interrupt requested. 1 This bit is set when the FlexCAN state changes to bus off. If the CANCTRL[BOFFMSK] bit is set an interrupt request is generated. This interrupt is not requested after reset.
1 ERRINT	Error interrupt. Indicates that at least one of the ERRSTAT[15:10] bits is set. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No error interrupt request. 1 At least one of the error bits is set. If the CANCTRL[ERRMSK] bit is set, an interrupt request is generated.
0	Reserved, must be cleared.

23.3.7   Interrupt Mask Register (IMASK)

IMASK contains one interrupt mask bit per buffer. It enables the CPU to determine which buffer generates an interrupt after a successful transmission/reception (when the corresponding IFLAG bit is set).

Address: 0xFC02_0028 (IMASK)   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									BUFnM							
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
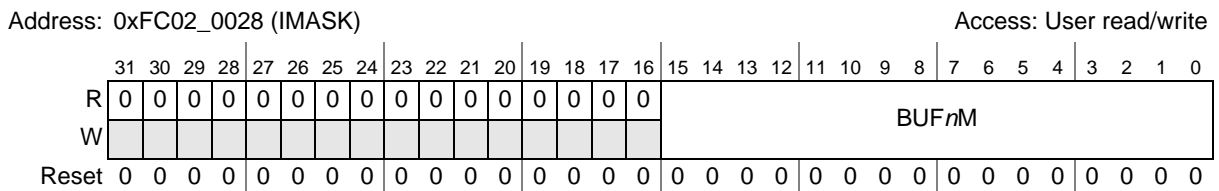
Figure 23-10. FlexCAN Interrupt Mask Register (IMASK)

Table 23-9. IMASK Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BUFnM	Buffer interrupt mask. Enables the respective FlexCAN message buffer (MB0 to MB15) interrupt. These bits allow the CPU to designate which buffers generate interrupts after successful transmission/reception. 0 The interrupt for the corresponding buffer is disabled. 1 The interrupt for the corresponding buffer is enabled. **Note:** Setting or clearing an IMASK bit can assert or negate an interrupt request, if the corresponding IFLAG bit it is set.

23.3.8   Interrupt Flag Register (IFLAG)

IFLAG contains one interrupt flag bit per buffer. Each successful transmission/reception sets the corresponding IFLAG bit and, if the corresponding IMASK bit is set, generates an interrupt.

The interrupt flag is cleared by writing a 1, while writing 0 has no effect.

Address: 0xFC02_0030 (IFLAG)   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	BUF*n*I
W																	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
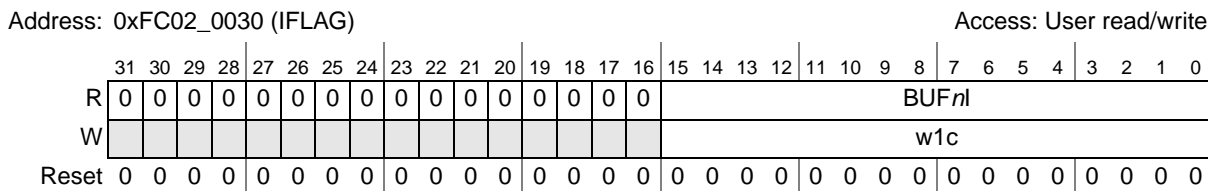
Figure 23-11. FlexCAN Interrupt Flags Register (IFLAG)

Table 23-10. IFLAG Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BUF*n*I	Buffer interrupt flag. Indicates a successful transmission/reception for the corresponding message buffer. If the corresponding IMASK bit is set, an interrupt request is generated. The user must write a 1 to clear an interrupt flag; writing 0 has no effect. 0 No such occurrence. 1 The corresponding buffer has successfully completed transmission or reception.

23.3.9   Message Buffer Structure

The message buffer memory map starts at an offset of 0x80 from the FlexCAN's base address (0xFC02_0000). The 256-byte message buffer space is fully used by the16 message buffer structures.

Each message buffer consists of a control and status field that configures the message buffer, an identifier field for frame identification, and up to 8 bytes of data.
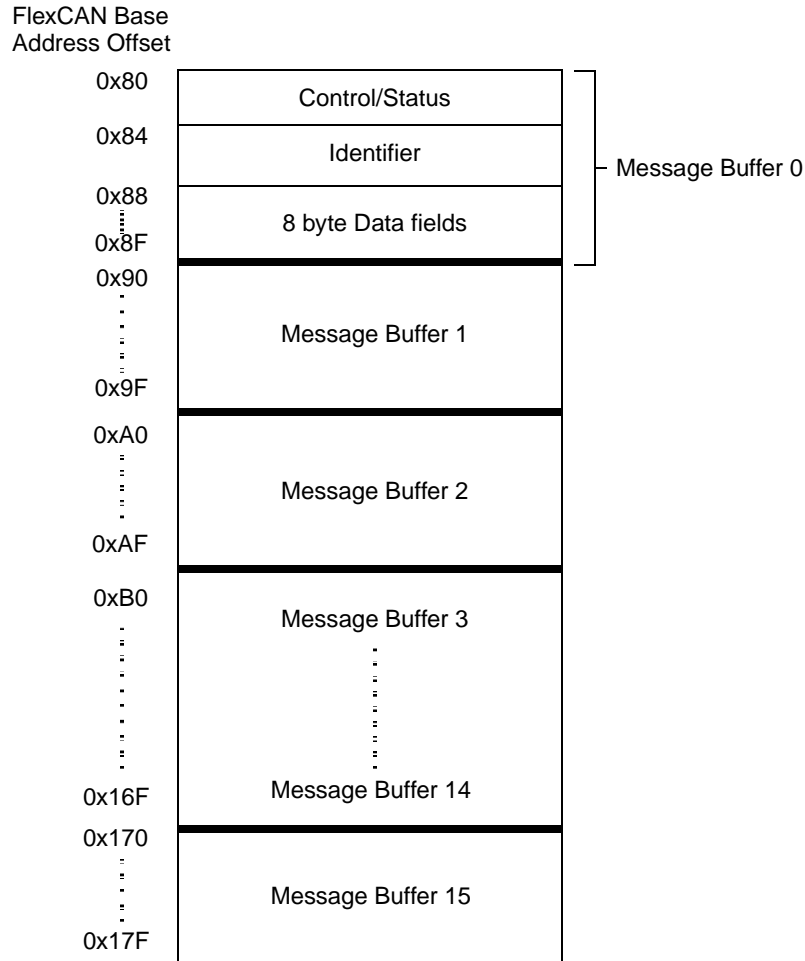
Figure 23-12. FlexCAN Message Buffer Memory Map

The message buffer structure used by the FlexCAN module is shown in Figure 23-13. Standard and extended frames used in the *CAN Specification Version 2.0, Part B* are represented. A standard frame is represented by the 11-bit standard identifier, and an extended frame is represented by the combined 29-bits of the standard identifier (11 bits) and the extended identifier (18 bits).
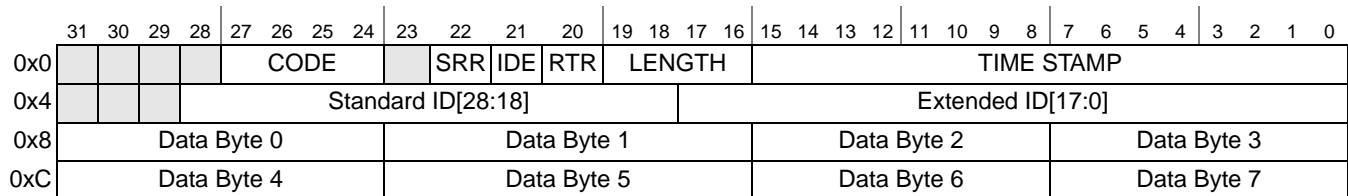
Figure 23-13. Message Buffer Structure for Extended and Standard Frames

Table 23-11. Message Buffer Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27–24 CODE	Message buffer code. Can be accessed (read or write) by the CPU and by the FlexCAN module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 23-12 and Table 23-13. See Section 23.3.10, "Functional Overview," for additional information.
23	Reserved, must be cleared.
22 SRR	Substitute remote request. Fixed recessive bit, used only in extended format. It must be set by the user for transmission (Tx Buffers) and is stored with the value received on the CAN bus for Rx receiving buffers. It can be received as recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in Extended Format frames 1 Recessive value is compulsory for transmission in Extended Format frames
21 IDE	ID extended bit. Identifies whether the frame format is standard or extended. 0 Standard frame format 1 Extended frame format
20 RTR	Remote transmission request. Used for requesting transmissions of a data frame. If FlexCAN transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a data frame to be transmitted 1 Indicates the current MB has a remote frame to be transmitted
19–16 LENGTH	Length of data in bytes. Indicates the length (in bytes) of the Rx or Tx data; data is located in offset 0x8 through 0xF of the MB space (see Figure 23-13). In reception, this field is written by the FlexCAN module, copied from the DLC (data length code) field of the received frame. DLC is defined by the *CAN Specification* and refers to the data length of the actual frame before it is copied into the message buffer. In transmission, this field is written by the CPU and is used as the DLC field value of the frame to be transmitted. When RTR is set, the frame to be transmitted is a remote frame and is transmitted without the DATA field, regardless of the LENGTH field.
15–0 TIME STAMP	Free-running counter time stamp. Stores the value of the free-running timer which is captured when the beginning of the identifier (ID) field appears on the CAN bus.
31–29	Reserved, must be cleared.
28–0 ID	Standard frame identifier: In standard frame format, only the 11 most significant bits (28 to 18) are used for frame identification in receive and transmit cases. The 18 least significant bits are ignored.
	Extended frame identifier: In extended frame format, all bits (the 11 bits of the standard frame identifier and the 18 bits of the extended frame identifier) are used for frame identification in receive and transmit cases.
31–24, 23–16, 15–8, 7–0 DATA	Data field. Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU provides the data to be transmitted within the frame.

Table 23-12. Message Buffer Code for Rx Buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active.	—	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the control & status (C/S) word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code remains FULL.
		0110	If the MB is FULL and a new frame should be written into this MB before the CPU had time to read it, the MB is overwritten, and the code is automatically updated to OVERRUN.
0110	OVERRUN: A frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB, the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB is overwritten again, and the code remains OVERRUN.
0XY1[1]	BUSY: Flexcan is updating the contents of the MB with a new receive frame. The CPU should not try to access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

[1] For transmit message buffers (see Table 23-13), the BUSY bit should be ignored upon read.

Table 23-13. Message Buffer Code for Tx Buffers

MB*n*[RTR]	Initial Tx Code	Code After Successful Transmission	Description
X	1000	—	INACTIVE: Message buffer not ready for transmit and participates in the arbitration process.
0	1100	1000	Data frame to be transmitted once, unconditionally. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Remote frame to be transmitted unconditionally once, and message buffer becomes an Rx message buffer with the same ID for data frames.

Table 23-13. Message Buffer Code for Tx Buffers (continued)

MB*n*[RTR]	Initial Tx Code	Code After Successful Transmission	Description
0	1010	1010	Transmit a data frame when a remote request frame with the same ID is received. This message buffer participates simultaneously in the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs, this message buffer is allowed to participate in the current arbitration process and the CODE field is automatically updated to 1110 to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the code automatically returns to 1010 to restart the process again.
0	1110	1010	This is an intermediate code automatically written to the message buffer as a result of match to a remote request frame. The data frame is transmitted unconditionally once, and then the code automatically returns to 1010. The CPU can also write this code with the same effect.

23.3.10 Functional Overview

The FlexCAN module is flexible in that each one of its 16 message buffers (MBs) can be assigned as a transmit buffer or a receive buffer. Each MB, which is up to 8 bytes long, is also assigned an interrupt flag bit that indicates successful completion of transmission or reception.

An arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID or the MB ordering. A matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. Data coherency mechanisms are implemented to guarantee data integrity during MB manipulation by the CPU.

Before proceeding with the functional description, an important concept must be explained. A message buffer is said to be active at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a 0000 code is inactive (refer to Table 23-12). Similarly, a Tx MB with a 1000 code is inactive (refer to Table 23-13). An MB not programmed with 0000 or 1000 is temporarily deactivated (does not participate in the current arbitration/matching run) when the CPU writes to the C/S field of that MB.

23.3.11 Transmit Process

The CPU prepares or changes an MB for transmission by writing the following:

1. Control/status word to hold Tx MB inactive (CODE = 1000)
2. ID word
3. Data bytes
4. Control/status word (active CODE, LENGTH)

NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or ID matching processes, giving time for the CPU to program the rest of the MB (see Section 23.3.15.2, "Message Buffer Deactivation"). After the MB is activated in the fourth step, it participates in the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the free running timer (TIMER) is written into the message buffer's time stamp field, the code field in the control and status word is updated, a status flag is set in the IFLAG register, and an interrupt is generated if allowed by the corresponding IMASK register bit. The new code field after transmission depends on the code that was used to activate the MB in step four (see Table 23-13).

23.3.12 Arbitration Process

The arbitration process is an algorithm executed by the message buffer management (MBM) that scans the entire MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers are scanned to find the lowest ID or the lowest MB number, depending on the CANCTRL[LBUF] bit.

NOTE

If CANCTRL[LBUF] is cleared, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in idle or bus off state and the CPU writes to the C/S word of any MB
- Upon leaving freeze mode

After the highest priority MB is selected, it is transferred to a temporary storage space called serial message buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called move-out. At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to 8 data bytes, even if the data length code (DLC) value is bigger. Refer to Section 23.3.15.1, "Serial Message Buffers (SMBs)," for more information on serial message buffers.

23.3.13 Receive Process

The CPU prepares or changes an MB for frame reception by writing the following:

1. Control/status word to hold Rx MB inactive (CODE = 0000)

2. ID word

3. Control/status word to mark the Rx MB as active and empty (CODE = 0100)

NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was a pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or matching process, giving time for the CPU to program the rest of the MB. After the MB is activated in the third step, it is able to receive CAN frames that match the programmed ID. At the end of a successful reception, the value of the free running timer (TIMER) is written into the time stamp field, the received ID, data (8 bytes at most) and length fields are stored, the CODE field in the control and status word is updated (see Table 23-12), and a status flag is set in the IFLAG register and an interrupt is generated if allowed by the corresponding IMASK bit.

The CPU should read a receive frame from its MB by reading the following:

1. Control/status word (mandatory—activates internal lock for this buffer)

2. ID (optional—needed only if a mask was used)

3. Data field words

4. Free-running timer (Releases internal lock —optional)

Upon reading the control and status word, if the BUSY bit is set in the CODE field, then the CPU should defer the access to the MB until this bit is negated. Reading the free running timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Only a single MB is locked at a time. The only mandatory CPU read operation is the one on the control and status word to assure data coherency.

The CPU should synchronize to frame reception by an IFLAG bit for the specific MB (see Section 23.3.8, "Interrupt Flag Register (IFLAG)"), and not by the control/status word CODE field for that MB. Polling the CODE field does not work because after a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the CODE field does not return to EMPTY. It remains FULL, as explained in Table 23-12. If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary, never do polling by directly reading the C/S word of the MBs. Instead, read the IFLAG register.

The received identifier field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking.

23.3.13.1 Self-Received Frames

Self-received frames are frames that are sent by the FlexCAN and received by itself. The FlexCAN sends a frame externally through the physical layer onto the CAN bus. If the ID of the frame matches the ID of the FlexCAN MB, the frame is received by the FlexCAN. Such a frame is a self-received frame. FlexCAN does not receive frames transmitted by itself if another device on the CAN bus has an ID that matches the FlexCAN Rx MB ID.

23.3.14 Matching Process

The matching process is an algorithm that scans the entire MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. Only MBs programmed to receive participate in the matching process for received frames.

While the ID, DLC and data fields are retrieved from the CAN bus, they are stored temporarily in the serial message buffer (Section 23.3.15.1, "Serial Message Buffers (SMBs)"). The matching process takes place during the CRC field. If a matching ID is found in one of the MBs, the contents of the SMB are transferred to the matched MB during the sixth bit of the end-of-frame field of the CAN protocol. This operation is called move-in. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

An MB with a matching ID is free to receive a new frame if the MB is not locked (see Section 23.3.15.3, "Locking and Releasing Message Buffers"). The CODE field is EMPTY, FULL, or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB).

Matching to a range of IDs is possible by using ID acceptance masks. FlexCAN supports a masking scheme with three mask registers (RXGMASK, RX14MASK, and RX15MASK). During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is don't care.

23.3.15 Message Buffer Managing

To maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Section 23.3.11, "Transmit Process" and Section 23.3.13, "Receive Process." Any form of CPU accessing a MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

23.3.15.1 Serial Message Buffers (SMBs)

To allow double buffering of messages, the FlexCAN has two shadow buffers called serial message buffers. These two buffers are used by the FlexCAN for buffering received messages and messages to be transmitted. Only one SMB is active at a time, and its function depends upon the operation of the FlexCAN at that time. At no time does the user have access to or visibility of these two buffers.

23.3.15.2 Message Buffer Deactivation

If the CPU wants to change the function of an active MB, the recommended procedure is to put the module into freeze mode and then change the CODE field of that MB. This is a safe procedure because the FlexCAN waits for pending CAN bus and MB moving activities to finish before entering freeze mode. Nevertheless, a mechanism is provided to maintain data coherence when the CPU writes to the control and status word of active MBs out of freeze mode.

Any CPU write access to the C/S word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. This mechanism is called MB deactivation. It is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent; therefore, that MB is deactivated.

Even with the coherence mechanism described above, writing to the C/S word of active MBs when not in freeze mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN continues looking for another matching MB within the ones it has not scanned yet. If it can not find one, the message is lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame is lost even if the second matching MB was free to receive.

- If a Tx MB containing the lowest ID is deactivated after the FlexCAN has scanned it, the FlexCAN looks for another winner within the MBs that it has not yet scanned. Therefore, it may transmit an MB that may not have the lowest ID at the time because a lower ID might be present that it had already scanned before the deactivation.

- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted, but no interrupt is issued and the CODE field is not updated.

23.3.15.3 Locking and Releasing Message Buffers

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the control and status word of an active not empty Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB.

The lock is released when the CPU reads the free running timer (global unlock operation), or when it reads the control and status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

NOTE

> The locking mechanism only applies to Rx MBs which have a code different than INACTIVE (0000) or EMPTY1 (0100). Also, Tx MBs can not be locked.

Suppose, for example, that FlexCAN has already received and stored a message into one of the MBs. Suppose now that the CPU decides to read that MB at the same time another message with the same ID is arriving. When the CPU reads the control and status word, the MB is locked. The new message arrives and the matching algorithm finds out that the matching MB is not free to receive It remains in the SMB waiting for the MB to be unlocked, and only then, is it written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there is no indication of lost messages in the code field of the MB or in the error and status register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the code field is set. If the CPU reads the control and status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is cleared.

If the BUSY bit is set or if the MB is empty, then reading the control and status word does not lock the MB.

NOTE

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated, and the MB is marked as invalid for the current matching round. Any pending message on the SMB is not transferred to the MB anymore.

23.3.16   CAN Protocol Related Frames

23.3.16.1   Remote Frames

The remote frame is a message frame transmitted to request a data frame. The FlexCAN can be configured to transmit a data frame automatically in response to a remote frame, or to transmit a remote frame and then wait for the responding data frame to be received.

When transmitting a remote frame, the user initializes a message buffer as a transmit message buffer with the RTR bit set. After this remote frame is transmitted successfully, the transmit message buffer automatically becomes a receive message buffer, with the same ID as the remote frame that was transmitted.

When a remote frame is received by the FlexCAN, the remote frame ID is compared to the IDs of all transmit message buffers programmed with a CODE of 1010. If there is an exact matching ID, the data frame in that message buffer is transmitted. If the RTR bit in the matching transmit message buffer is set, the FlexCAN transmits a remote frame as a response.

A received remote frame is not stored in a receive message buffer. It is only used to trigger the automatic transmission of a frame in response. The mask registers are not used in remote frame ID matching. All ID bits (except RTR) of the incoming received frame must match for the remote frame to trigger a response transmission. The matching message buffer immediately enters the internal arbitration process, but is considered as a normal Tx MB, with no higher priority. The data length of this frame is independent of the data length code (DLC) field in the remote frame that initiated its transmission.

23.3.16.2   Overload Frames

Overload frame transmissions are not initiated by the FlexCAN unless certain conditions are detected on the CAN bus. These conditions include detection of a dominant bit in the following:

- First or second bit of intermission
- Seventh (last) bit of the end-of-frame (EOF) field in receive frames
- Eighth (last) bit of the error frame delimiter or overload frame delimiter

23.3.17 Time Stamp

The value of TIMER is sampled at the beginning of the identifier field on the CAN bus. For a message being received, the time stamp is stored in the TIMESTAMP entry of the receive message buffer at the time the message is written into that buffer. For a message being transmitted, the TIMESTAMP entry is written into the transmit message buffer after the transmission has completed successfully.

The free-running timer can optionally be reset upon the reception of a frame into message buffer 0. This feature allows network time synchronization to be performed. See the CANCTRL[TSYN] bit.

23.3.18 Bit Timing

The FlexCAN module CANCTRL register configures the bit timing parameters required by the CAN protocol. The CLK_SRC, PRESDIV, RJW, PSEG1, PSEG2, and the PROPSEG fields allow the user to configure the bit timing parameters.

The CANCTRL[CLK_SRC] bit defines whether the module uses the internal bus clock or the output of the crystal oscillator via the EXTAL pin. The crystal oscillator clock should be selected when a tight tolerance (up to 0.1%) is required for the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks. The value of this bit should not be changed, unless the module is in disable mode (CANMCR[MDIS] bit is set)

The PRESDIV field controls a prescaler that generates the serial clock (S-clock), whose period defines the time quantum used to compose the CAN waveform. A time quantum is the atomic unit of time managed by the CAN engine.
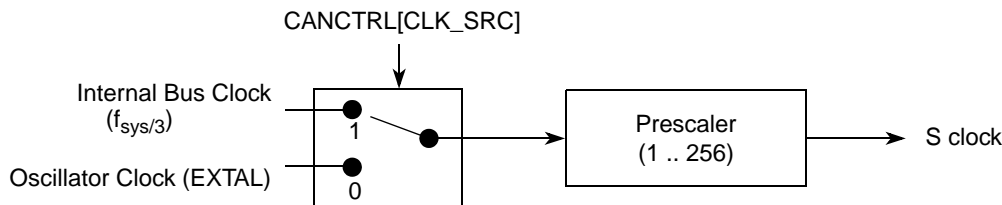
Figure 23-14. CAN Engine Clocking Scheme

$$f_{Tq} = \frac{f_{sys/3} \text{ or EXTAL}}{(PRESDIV + 1)}$$ *Eqn. 23-6*

A bit time is subdivided into three segments[1] (see Figure 23-15 and Table 23-14):

- SYNC_SEG: Has a fixed length of one time quantum. Signal edges are expected to happen within this section.

- Time Segment 1: Includes the propagation segment and the phase segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CANCTRL register so that their sum (plus 2) is in the range of 4 to 16 time quanta.

- Time Segment 2: Represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CANCTRL register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})}$$ *Eqn. 23-7*

1. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.
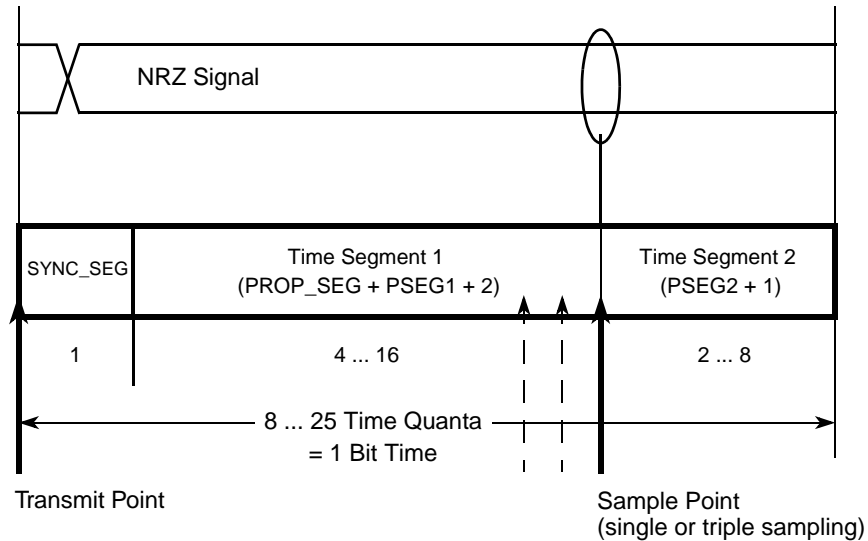
Figure 23-15. Segments within the Bit Time

Table 23-14. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 23-15 gives an overview of the CAN compliant segment settings and the related parameter values.

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module

Table 23-15. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4

Table 23-15. CAN Standard Compliant Bit Time Segment Settings (continued)

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

23.4 Initialization/Application Information

Initialization of the FlexCAN includes the initial configuration of the message buffers and configuration of the CAN communication parameters following a reset, as well as any reconfiguration that may be required during operation. The FlexCAN module may be reset in three ways:

- Device level hard reset—resets all memory mapped registers asynchronously
- Device level soft reset—resets some of the memory mapped registers synchronously (refer to Table 23-1 to see which registers are affected by soft reset)
- CANMCR[SOFT_RST] bit—has the same effect as the device level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The CANMCR[SOFT_RST] bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source, CANCTRL[CLK_SRC], should be selected while the module is in disable mode. After the clock source is selected and the module is enabled (CANMCR[MDIS] bit cleared), the FlexCAN automatically enters freeze mode. In freeze mode, the FlexCAN is un-synchronized to the CAN bus, the CANMCR register's HALT and FRZ bits are set, the internal state machines are disabled, and the CANMCR register's FRZ_ACK and NOT_RDY bits are set. The CANTX pin is in recessive state and the FlexCAN does not initiate any transmission or reception of CAN frames. The message buffers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, the FlexCAN must be in freeze mode (see Section 23.1.3.2, "Freeze Mode"). The following is a generic initialization sequence applicable to the FlexCAN module:

1. Initialize all operation modes in the CANCTRL register.
   a) Initialize the bit timing parameters PROPSEG, PSEGS1, PSEG2, and RJW.
   b) Select the S-clock rate by programming the PRESDIV field.
   c) Select the internal arbitration mode via the LBUF bit.
2. Initialize message buffers.
   a) The control/status word of all message buffers must be written as an active or inactive message buffer.
   b) All other entries in each message buffer should be initialized as required.
3. Initialize RXGMASK, RX14MASK, and RX15MASK registers for acceptance mask as needed.

4. Initialize FlexCAN interrupt handler.

    a) Initialize the interrupt controller registers for any needed interrupts. See Chapter 14, "Interrupt Controller Modules," for more information.

    b) Set the required mask bits in the IMASK register (for all message buffer interrupts) and the CANCTRL (for bus off and error interrupts).

5. Clear the CANMCR[HALT] bit. At this point, the FlexCAN attempts to synchronize with the CAN bus.

23.4.1 Interrupts

There are 19 interrupt sources for the FlexCAN module. An interrupt for each of the 16 MBs. Plus, a combined interrupt for all 16 MBs is generated by logically OR'ing all the interrupt sources from the MBs. In this case, the CPU must read the IFLAGn register to determine which MB caused the interrupt. The other interrupt sources (bus off and error) act in the same manner, and are located in the ERRSTAT register. The bus off and error interrupt mask bits are located in the CANCTRL register.

Chapter 24
Synchronous Serial Interface (SSI)

24.1 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 24-1, 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 9, "Chip Configuration Module (CCM)," for detailed information on these bit fields.
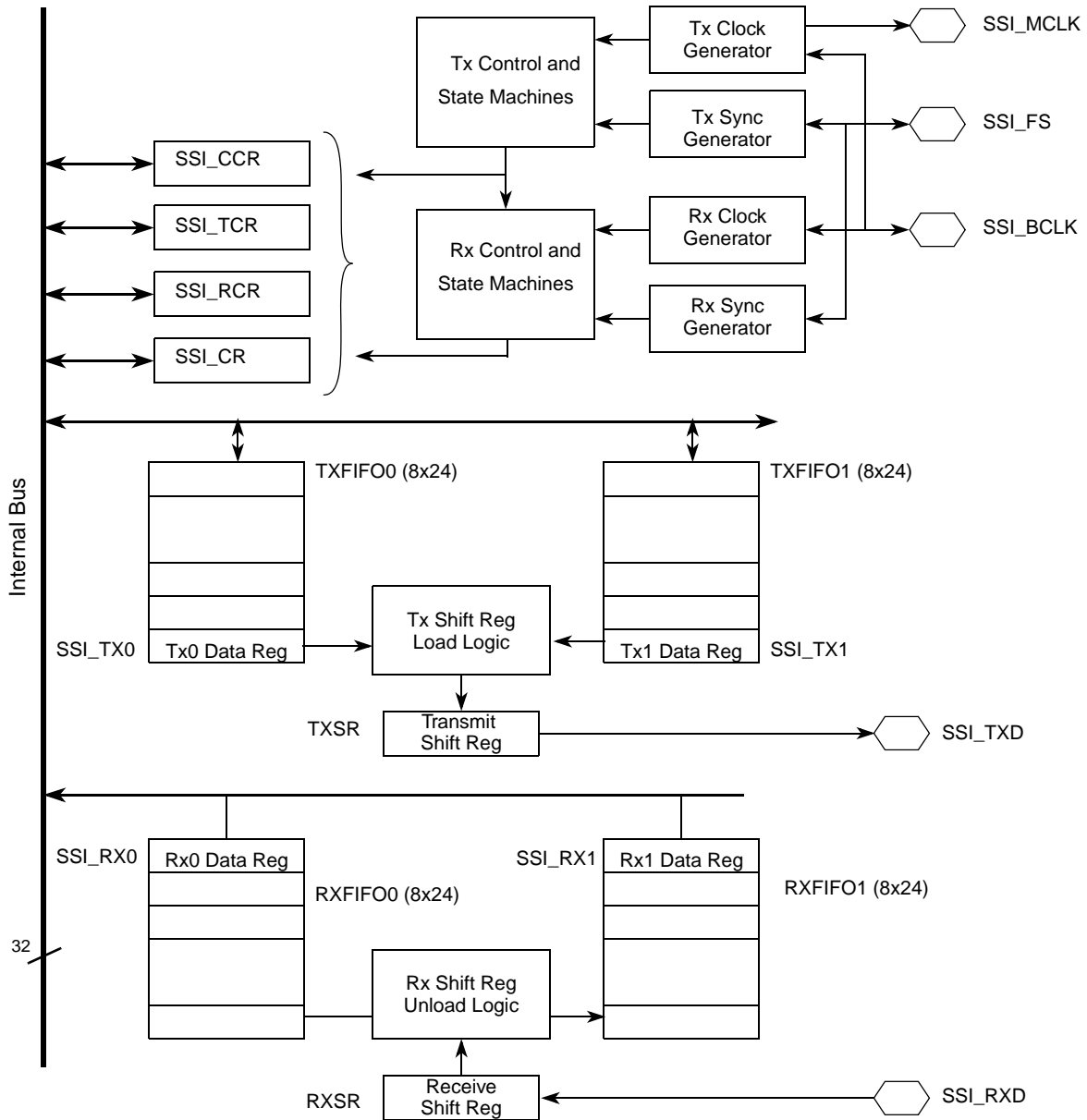
Figure 24-1. SSI Block Diagram

24.1.1   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 GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 13, "General Purpose I/O Module") prior to configuring the SSI.

24.1.2 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 of the four FIFOs is 8x24 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

24.1.3 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 24-1 lists these operating modes and some of the typical applications in which they can be used:

Table 24-1. 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^2S mode
- AC97 mode
  - — AC97 fixed mode
  - — AC97 variable mode

In non-I^2S 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^2S slave mode, the SSI's programmed word length setting can be lesser than or equal to the word length setting of the I^2S 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 24.4.1, "Detailed Operating Mode Descriptions," for more details on the above modes.

24.2 External Signal Description

The five SSI signals are explained below.

Table 24-2. 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

24.2.1 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 9, "Chip Configuration Module (CCM)," and Figure 24-37.

24.2.2 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.

24.2.3 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.

24.2.4 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.

24.2.5 SSI_RXD — Serial Receive Data

The SSI_RXD port is an input and brings serial data into the receive data shift register.

24.2.6   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 24-2 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).

SSI_TXD
SSI_RXD
SSI_BCLK
SSI_FS
SSI_MCLK

SSI Internal Continuous Clock
(RXDIR=0, TXDIR=1, TFDIR=1, SYN=1, MCE=1)

SSI I2S Master Mode
(I2S_Mode=01, MCE)

SSI_TXD
SSI_RXD
SSI_BCLK
SSI_FS

SSI External Continuous Clock
(RXDIR=0, TXDIR=0, TFDIR=0, SYN=1)

SSI I2S Slave Mode
(I2S_Mode=10)

SSI_TXD
SSI_RXD
SSI_BCLK

SSI Internal Gated Clock
(RXDIR=1, TXDIR=1, SYN=1)

SSI_TXD
SSI_RXD
SSI_BCLK

SSI External Gated Clock
(RXDIR=1, TXDIR=0, SYN=1)

Figure 24-2. Synchronous SSI Configurations—Continuous and Gated Clock

Figure 24-3 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.
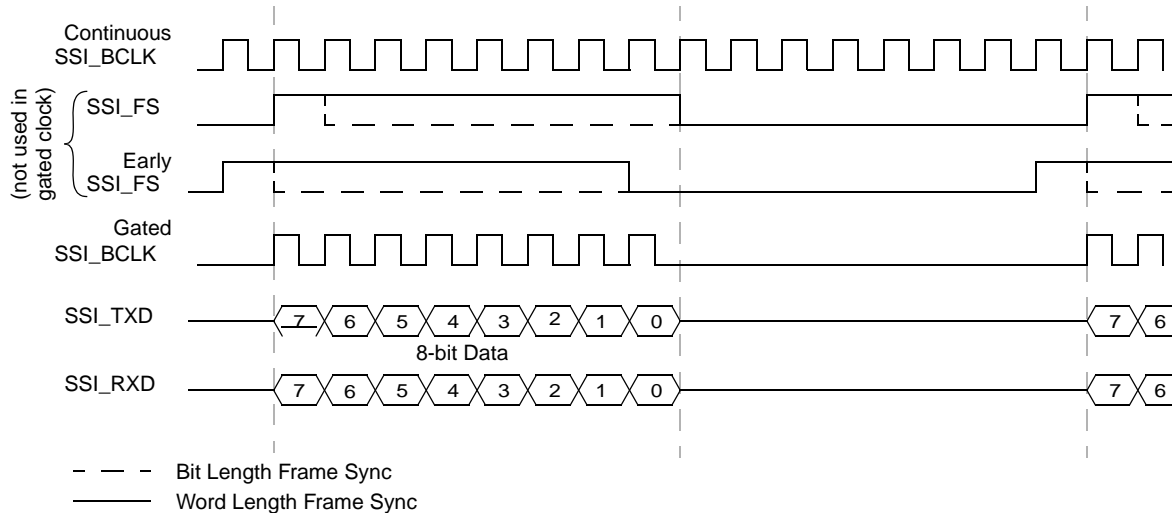
Figure 24-3. Serial Clock and Frame Sync Timing

Table 24-3. 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	—

24.3   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 24-4. 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	24.3.1/24-8
0xFC0B_C004	SSI Transmit Data Register 1 (SSI_TX1)	32	R/W	0x0000_0000	24.3.1/24-8
0xFC0B_C008	SSI Receive Data Register 0 (SSI_RX0)	32	R	0x0000_0000	24.3.4/24-10
0xFC0B_C00C	SSI Receive Data Register 1 (SSI_RX1)	32	R	0x0000_0000	24.3.4/24-10
0xFC0B_C010	SSI Control Register (SSI_CR)	32	R/W	0x0000_0000	24.3.7/24-13
0xFC0B_C014	SSI Interrupt Status Register (SSI_ISR)	32	R	0x0000_3003	24.3.8/24-15

Table 24-4. 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	24.3.9/24-20
0xFC0B_C01C	SSI Transmit Configuration Register (SSI_TCR)	32	R/W	0x0000_0200	24.3.10/24-21
0xFC0B_C020	SSI Receive Configuration Register (SSI_RCR)	32	R/W	0x0000_0200	24.3.11/24-23
0xFC0B_C024	SSI Clock Control Register (SSI_CCR)	32	R/W	0x0004_0000	24.3.12/24-24
0xFC0B_C02C	SSI FIFO Control/Status Register (SSI_FCSR)	32	R/W	0x0081_0081	24.3.13/24-25
0xFC0B_C038	SSI AC97 Control Register (SSI_ACR)	32	R/W	0x0000_0000	24.3.14/24-27
0xFC0B_C03C	SSI AC97 Command Address Register (SSI_ACADD)	32	R/W	0x0000_0000	24.3.15/24-28
0xFC0B_C040	SSI AC97 Command Data Register (SSI_ACDAT)	32	R/W	0x0000_0000	24.3.16/24-29
0xFC0B_C044	SSI AC97 Tag Register (SSI_ATAG)	32	R/W	0x0000_0000	24.3.17/24-29
0xFC0B_C048	SSI Transmit Time Slot Mask Register (SSI_TMASK)	32	R/W	0x0000_0000	24.3.18/24-30
0xFC0B_C04C	SSI Receive Time Slot Mask Register (SSI_RMASK)	32	R/W	0x0000_0000	24.3.19/24-30

24.3.1 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 24.4.4, "Supported Data Alignment Formats."

Address: 0xFC0B_C000 (SSI_TX0) Access: User read/write
0xFC0B_C004 (SSI_TX1)

	31 30 29 28	27 26 25 24	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				SSI_TX				
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
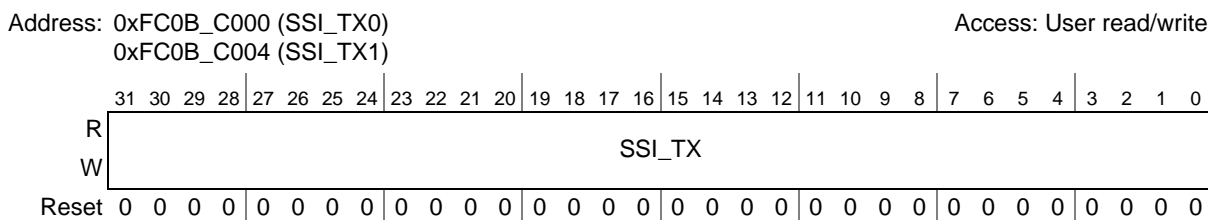
Figure 24-4. SSI Transmit Data Registers (SSI_TX0, SSI_TX1)

Table 24-5. 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 – 9 to SSI_TX0, Data9 does not overwrite Data1. Data1 – 8 are stored in the FIFO while Data9 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

24.3.2 SSI Transmit FIFO 0 and 1 Registers

The SSI transmit FIFO registers are 8x32-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.

24.3.3 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.
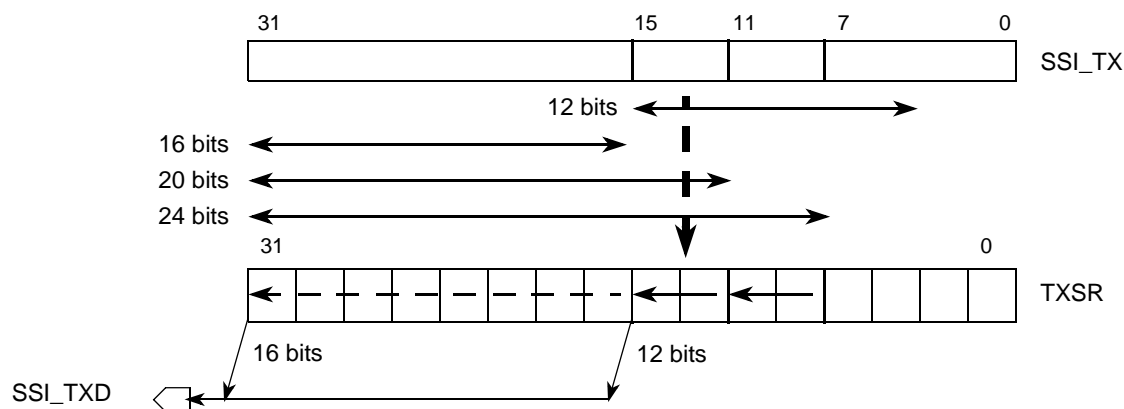
Figure 24-5. Transmit Data Path (TXBIT0=0, TSHFD=0) (msb Alignment)
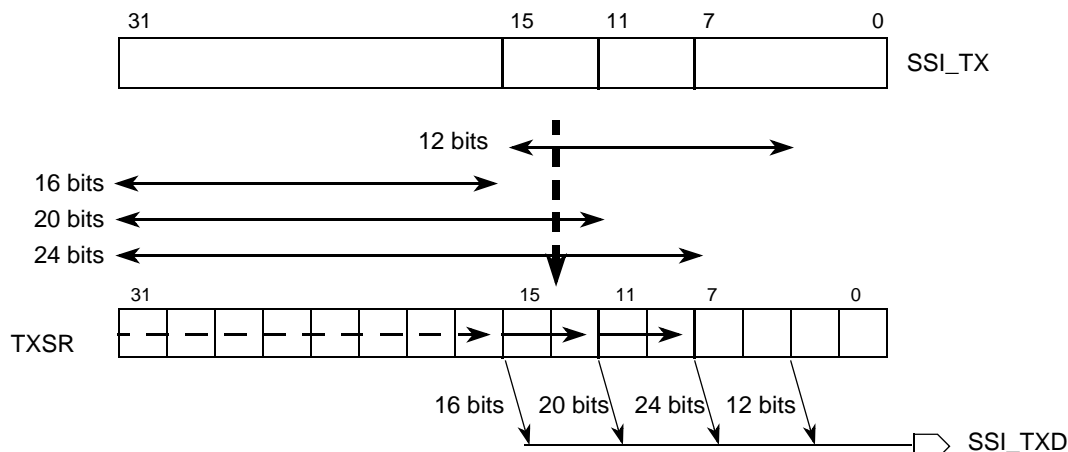
Figure 24-6. Transmit Data Path (TXBIT0=0, TSHFD=1) (msb Alignment)
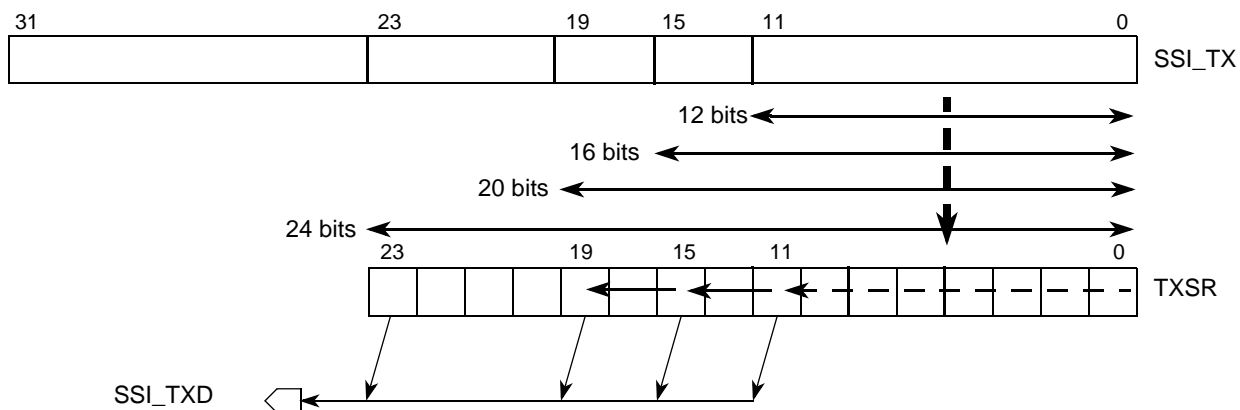
Figure 24-7. Transmit Data Path (TXBIT0=1, TSHFD=0) (lsb Alignment)
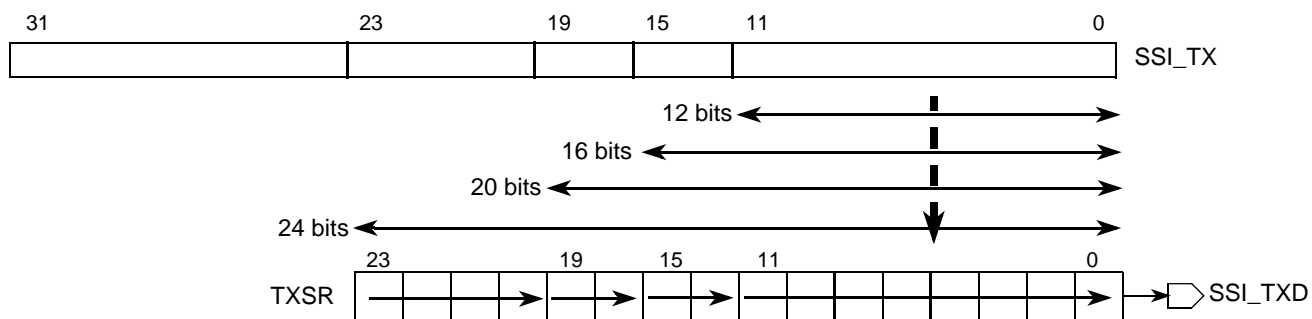
Figure 24-8. Transmit Data Path (TXBIT0=1, TSHFD=1) (lsb Alignment)

24.3.4 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 24.3.6, "SSI Receive Shift Register (RXSR)."

Address: 0xFC0B_C008 (SSI_RX0)   Access: User read/write
0xFC0B_C00C (SSI_RX1)

	31	30	29	28	27	26	25	24	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
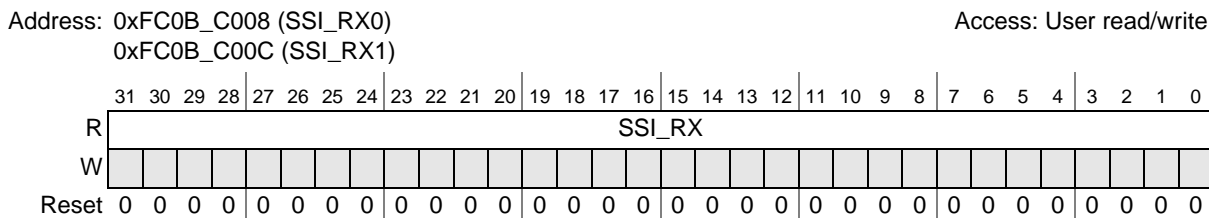
Figure 24-9. SSI Receive Data Registers (SSI_RX0, SSI_RX1)

Table 24-6. 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.

24.3.5 SSI Receive FIFO 0 and 1 Registers

The SSI receive FIFO registers are 8x32-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.

24.3.6 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.
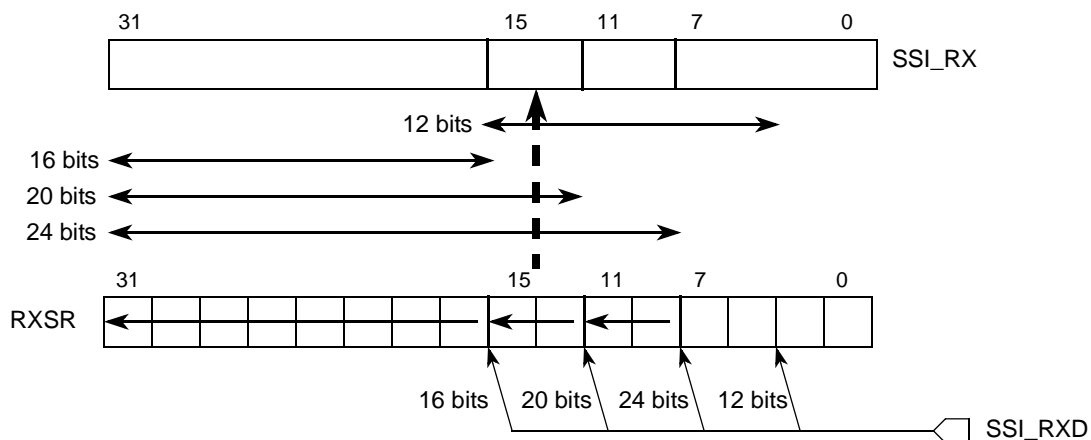
Figure 24-10. Receive Data Path (RXBIT0=0, RSHFD=0) (msb Alignment)
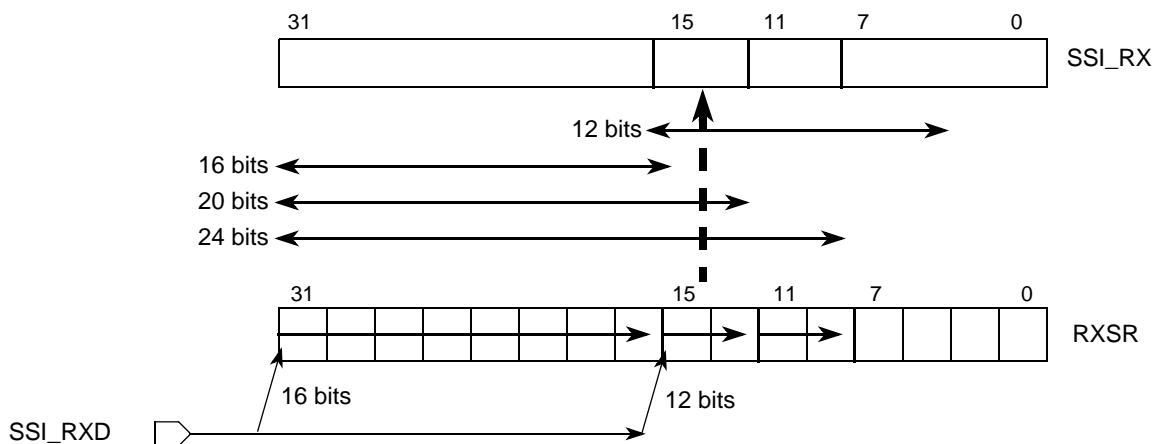
Figure 24-11. Receive Data Path (RXBIT0=0, RSHFD=1) (msb Alignment)
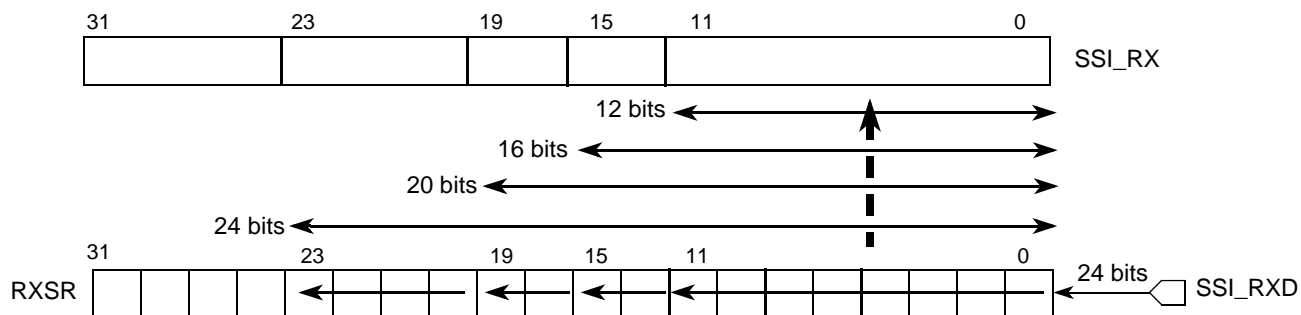
Figure 24-12. Receive Data Path (RXBIT0=1, RSHFD=0) (lsb Alignment)
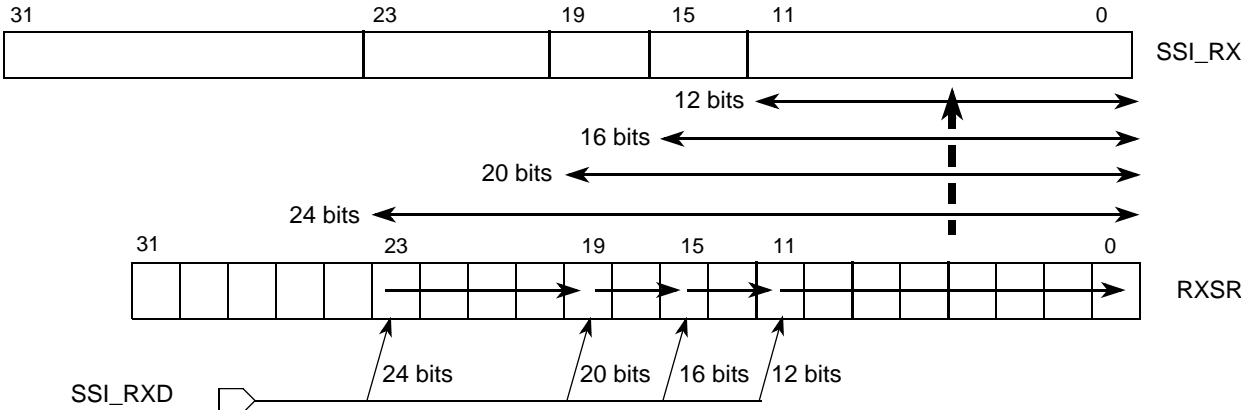
Figure 24-13. Receive Data Path (RXBIT0=1, RSHFD=1) (lsb Alignment)

24.3.7 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

Figure 24-14. SSI Control Register (SSI_CR)

Table 24-7. 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 24-7. 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^2S 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^2S mode select. Selects normal, I^2S master, or I^2S slave mode. Refer to Section 24.4.1.4, "I2S Mode," for a detailed description of I^2S mode. 00 Normal mode 01 I^2S master mode 10 I^2S 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

24.3.8 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 24.4.5, "Receive Interrupt Enable Bit Description," and Section 24.4.6, "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 24-15. SSI Interrupt Status Register (SSI_ISR)

Table 24-8. 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 24-8. SSI_ISR Field Descriptions (continued)

Field	Description
15 RDR1	Receive data ready 1. Only valid in two-channel mode. Indicates new data is available for the processor to read.

| Rx FIFO1 | Receive data 1 interrupt | |
	Required conditions	Trigger
Enabled	• SSI_IER[RIE] set • SSI_IER[RFF1] set	• SSI_ISR[RFF1] sets
Disabled	• SSI_IER[RIE] set • SSI_IER[RDR1] set	• SSI_RX1 loaded with new value

Rx FIFO1	RDR1 is set when	RDR1 is cleared during any of the following
Enabled	• Rx FIFO1 loaded with new value	• Rx FIFO1 is empty • SSI reset • POR reset
Disabled	• SSI_RX1 loaded with new value	• SSI_RX1 is read • SSI reset • POR reset

Field	Description
14 RDR0	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
13 TDE1	Transmit data register empty 1. Only valid in two-channel mode. Indicates that data needs to be written to the SSI.

| Tx FIFO1 | Transmit data 1 interrupt | |
	Required conditions	Trigger
Enabled	• SSI_IER[TIE] set • SSI_IER[TDE1] set	• SSI_ISR[TDE1] sets
Disabled	• SSI_IER[TIE] set • SSI_IER[TDE1] set	• SSI_TX1 data transferred to TXSR

Tx FIFO1	TDE1 is set when	TDE1 is cleared when any of the following occur
Enabled	• At least one empty slot in Tx FIFO1	• Tx FIFO1 is full • SSI reset • POR reset
Disabled	• SSI_TX1 data transferred to TXSR	• SSI_TX1 is written • SSI reset • POR reset

Table 24-8. SSI_ISR Field Descriptions (continued)

Field	Description
12 TDE0	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
11 ROE1	Receiver overrun error 1. Only valid in two-channel mode. Indicates an overrun error has occurred. <table><tr><td rowspan="2">**Rx FIFO1**</td><td colspan="2">**Receiver overrun error 1 interrupt**</td></tr><tr><td>**Required conditions**</td><td>**Trigger**</td></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><td>**Rx FIFO1**</td><td>**ROE1 is set when all of the following occur**</td><td>**ROE1 is cleared when any of the following occur**</td></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> **Note:** If Rx FIFO 1 is enabled, the RFF1 flag indiates the FIFO is full. If Rx FIFO 1 is disabled, the RDR1 flag indicates the SSI_RX1 register is full.
10 ROE0	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
9 TUE1	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><td rowspan="2">**Tx FIFO1**</td><td colspan="2">**Transmit underrun error 1 interrupt**</td></tr><tr><td>**Required conditions**</td><td>**Trigger**</td></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><td>**Tx FIFO1**</td><td>**TUE1 is set when all of the following occur**</td><td>**TUE1 is cleared when any of the following occur**</td></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>

Table 24-8. SSI_ISR Field Descriptions (continued)

Field	Description
8 TUE0	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
7 TFS	Transmit frame sync. Indicates occurrence of a transmit frame sync during transmission of the last word written to the SSI_TX registers |SSI Mode|Transmit frame sync interrupt|
---	---	---
	Required conditions	Trigger
Normal	• SSI_IER[TIE] set • SSI_IER[TFS] set	• SSI_ISR[TFS] sets
Network				 	SSI Mode	TFS is set when	TFS is cleared when any of the following occur
---	---	---
Normal	• TFS is always set	• SSI reset • POR reset
Network	• First time slot transmission	• Starts transmitting next time slot • SSI reset • POR reset	 **Note:** 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).
6 RFS	Receive frame sync. Indicates occurrence of a receive frame sync during reception of the next word in SSI_RX registers. |SSI Mode|Receive frame sync interrupt|
---	---	---
	Required conditions	Trigger
Normal	• SSI_IER[RIE] set • SSI_IER[RFS] set	• SSI_ISR[RFS] sets
Network				 	SSI Mode	RFS is set when	RFS is cleared when any of the following occur
---	---	---
Normal	• RFS is always set	• SSI reset • POR reset
Network	• First time slot received	• Starts receiving next time slot • SSI reset • POR reset	

Table 24-8. SSI_ISR Field Descriptions (continued)

Field	Description
5 TLS 4 RLS	Transmit/receive last time slot. Indicates the current time slot is the last time slot of the frame. _Last time slot interrupts_ 		Required conditions	Trigger
---	---	---
TLS	• SSI_IER[TIE] set • SSI_IER[TLS] set	• SSI_ISR[TLS] sets
RLS	• SSI_IER[RIE] set • SSI_IER[RLS] set	• SSI_ISR[RLS] sets | |	Is set when	Is cleared when any of the following occur
---	---	---
TLS	• Start of last transmit time slot	• SSI_ISR is read with TLS set • SSI reset • POR reset
RLS	• End of last receive time slot	• SSI_ISR is read with RLS set • SSI reset • POR reset	
3 RFF1	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. 	Rx FIFO1	Receive FIFO full 1 interrupt	
---	---	---
	Required conditions	Trigger
Enabled	• SSI_IER[RIE] set • SSI_IER[RFF1] set	• SSI_ISR[RFF1] sets | | Rx FIFO1	RFF1 is set when	RFF1 is cleared when any of the following occur
---	---	---
Enabled	• Rx FIFO 1 level reaches Rx FIFO watermark 1 (RFWM1) threshold	• Rx FIFO 1 level falls below RFWM1 level • SSI reset • POR reset	
2 RFF0	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

Table 24-8. SSI_ISR Field Descriptions (continued)

Field	Description
1 TFE1	Transmit FIFO empty 1. Only valid when in two-channel mode and Tx FIFO 1 is enabled. _Table A_ _Table B_
0 TFE0	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

Table A:

Tx FIFO1	Transmit FIFO full 1 interrupt	
	Required conditions	**Trigger**
Enabled	• SSI_IER[RIE] set • SSI_IER[TFE1] set	• SSI_ISR[TFE1] sets

Table B:

Tx FIFO1	**TFE1 is set when any of the following occur**	**TFE1 is cleared when any of the following occur**
Enabled	• Tx FIFO 1 level falls below Tx FIFO watermark 1 (TFWM1) threshold • SSI reset • POR reset	• Tx FIFO 1 level is more than TFWM1 level

24.3.9 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 W	RDR1	RDR0	TDE1	TDE0	ROE1	ROE0	TUE1	TUE0	TFS	RFS	TLS	RLS	RFF1	RFF0	TFE1	TFE0
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 24-16. SSI Interrupt Enable Register (SSI_IER)

Table 24-9. 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 24.4.5, "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 24.4.6, "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 24.3.8, "SSI Interrupt Status Register (SSI_ISR)," for details on the individual bits. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.

24.3.10 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.

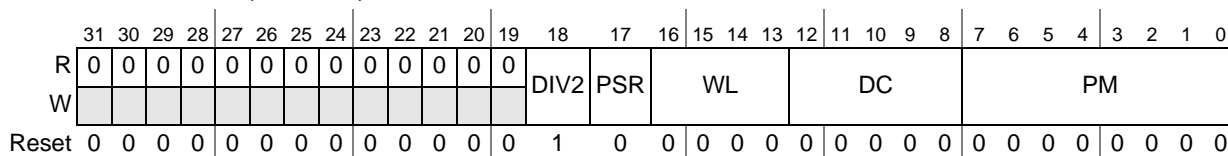Address: 0xFC0B_C01C (SSI_TCR) Access: User read/write

	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 24-17. SSI Transmit Configuration Register (SSI_TCR)

Table 24-10. 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 24-3 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

24.3.11 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 24-18. SSI Receive Configuration Register (SSI_RCR)

Table 24-11. 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 24-11. 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.

24.3.12 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 24-19. SSI Clock Control Register (SSI_CCR)

Table 24-12. 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 24-12. SSI_CCR Field Descriptions (continued)

Field	Description
16–13 WL	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 I²S 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 $2 \times (WL + 1)$. Refer to the below table for details of data word lengths supported by the SSI module. **Note:** In AC97 mode, if WL is set to any value other than 16 bits, a word length of 20 bits is used.

WL	Bits/word	Supported?	WL	Bits/word	Supported?
0000	2	No	1000	18	Yes
0001	4	No	1001	20	Yes
0010	6	No	1010	22	Yes
0011	8	Yes	1011	24	Yes
0100	10	Yes	1100	26	No
0101	12	Yes	1101	28	No
0110	14	No	1110	30	No
0111	16	Yes	1111	32	No

Field	Description
12–8 DC	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.
7–0 PM	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 24.4.2.2, "DIV2, PSR and PM Bit Description," for details regarding settings.

24.3.13 SSI FIFO Control/Status Register (SSI_FCSR)

Address: 0xFC0B_C02C (SSI_FCSR)   Access: User read/write

Figure 24-20. SSI FIFO Control/Status Register (SSI_FCSR)

Table 24-13. SSI_FCSR Field Descriptions

Field	Description
31–28 RFCNT1	Receive FIFO counter 1. Indicates the number of data words in receive FIFO 1. 0000 0 data words in receive FIFO. ... 1000 8 data words in receive FIFO. Else Reserved.
27–24 TFCNT1	Transmit FIFO counter 1. Indicates the number of data words in transmit FIFO 1. 0000 0 data words in transmit FIFO. ... 1000 8 data words in transmit FIFO. Else Reserved.
23–20 RFWM1	Receive FIFO full watermark 1. Controls threshold for when the SSI_ISR[RFF1] flag is set. RFF1 is set when the data level in Rx FIFO 1 reaches the selected threshold. 0001 RFF1 set when at least one data word has been written to the receive FIFO (RFCNT equals 0x1 – 0x8) ... 1000 RFF1 set when 8 data words have been written to the receive FIFO (RFCNT equals 0x8) Else Reserved.
19–16 TFWM1	Transmit FIFO empty watermark 1. Controls the threshold at which the SSI_ISR[TFE1] flag is set. The TFE1 flag is set when the data level in Tx FIFO 1 falls below the selected threshold. 0001 TFE1 set when there are greater than or equal to one empty slots in the Tx FIFO (TFCNT equals 0x0 – 0x7) ... 1000 TFE1 set when there are eight empty slots in the transmit FIFO (TFCNT equals 0x0)
15–12 RFCNT0	Receive FIFO counter 0. Indicates the number of data words in receive FIFO 0. See RFCNT1 for bit settings.
11–8 TFCNT0	Transmit FIFO counter 0. Indicates the number of data words in transmit FIFO 0. See TFCNT1 for bit settings.
7–4 RFWM0	Receive FIFO full watermark 0. Controls threshold for when the SSI_ISR[RFF0] flag is set. RFF0 is set when the data level in Rx FIFO 0 reaches the selected threshold. See RFWM1 for bit settings.
3–0 TFWM0	Transmit FIFO empty watermark 0. Controls the threshold for when the SSI_ISR[TFE0] flag is set. TFE0 is set when the data level in Tx FIFO 0 falls below the selected threshold. See TFWM1 for bit settings.

The following table indicates the status of the SSI_ISR[TFE0/1] flag, with different settings of the SSI_FCSR[TFWM0/1] bits and varying amounts of data in the Tx FIFO.

Table 24-14. Status of Transmit FIFO Empty Flag (SSI_ISR[TFE*n*])

Transmit FIFO Watermark (TFWM*n*)	Number of data in the Tx FIFO								
	0	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	0	0	0
4	1	1	1	1	1	0	0	0	0
5	1	1	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0	0
7	1	1	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0

24.3.14 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
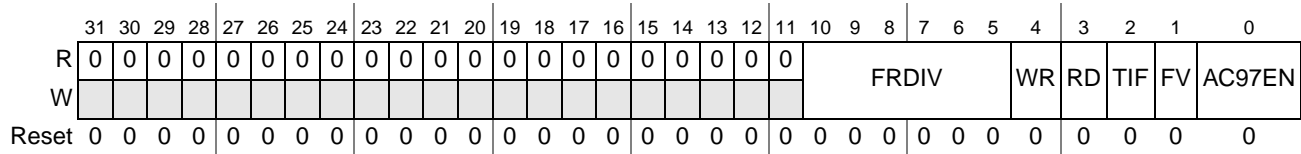
Figure 24-21. SSI AC97 Control Register (SSI_ACR)

Table 24-15. 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.

Table 24-15. SSI_ACR Field Descriptions (continued)

Field	Description
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 24.4.1.5, "AC97 Mode," for details of AC97 operation. 0 AC97 mode disabled 1 AC97 mode enabled

24.3.15 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
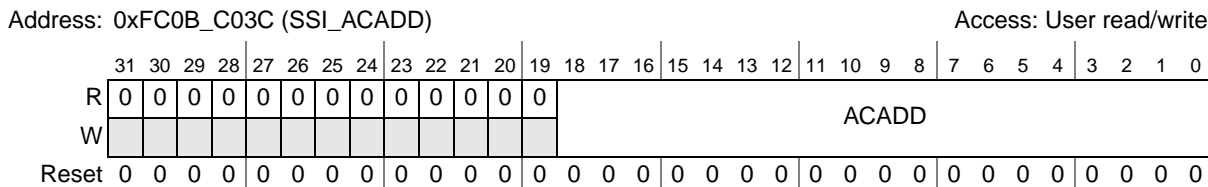
Figure 24-22. SSI AC97 Command Address Register (SSI_ACADD)

Table 24-16. 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.

24.3.16 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
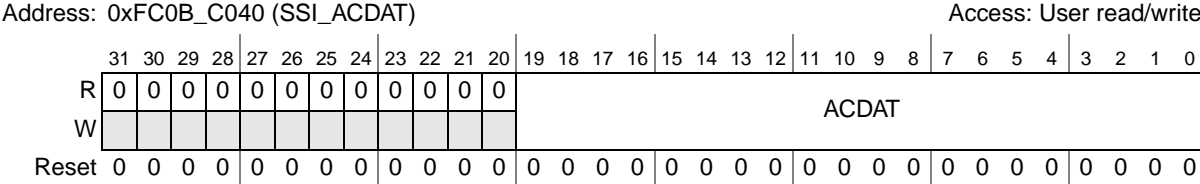
Figure 24-23. SSI AC97 Command Data Register (SSI_ACDAT)

Table 24-17. 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.

24.3.17 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
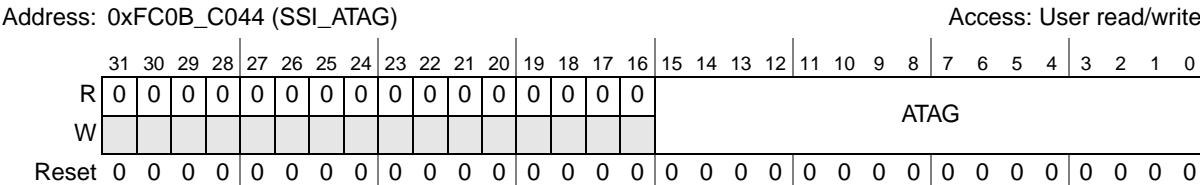
Figure 24-24. SSI AC97 Tag Register (SSI_ATAG)

Table 24-18. 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.

24.3.18 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
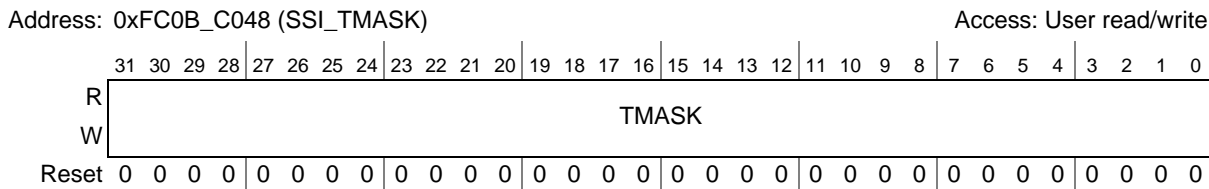
Figure 24-25. SSI Transmit Time Slot Mask Register (SSI_TMASK)

Table 24-19. 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^{2}S slave mode. 0 Valid time slot 1 Time slot masked (no data transmitted in this time slot)

24.3.19 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					RMASK			
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
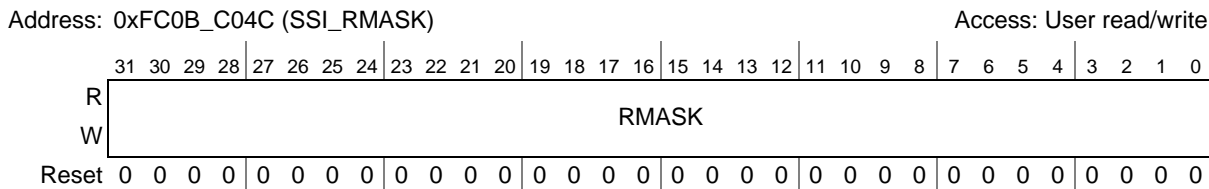
Figure 24-26. SSI Receive Time Slot Mask Register (SSI_RMASK)

Table 24-20. 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)

24.4 Functional Description

24.4.1 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.

24.4.1.1 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.

24.4.1.1.1 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.

24.4.1.1.2 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 24-27 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.
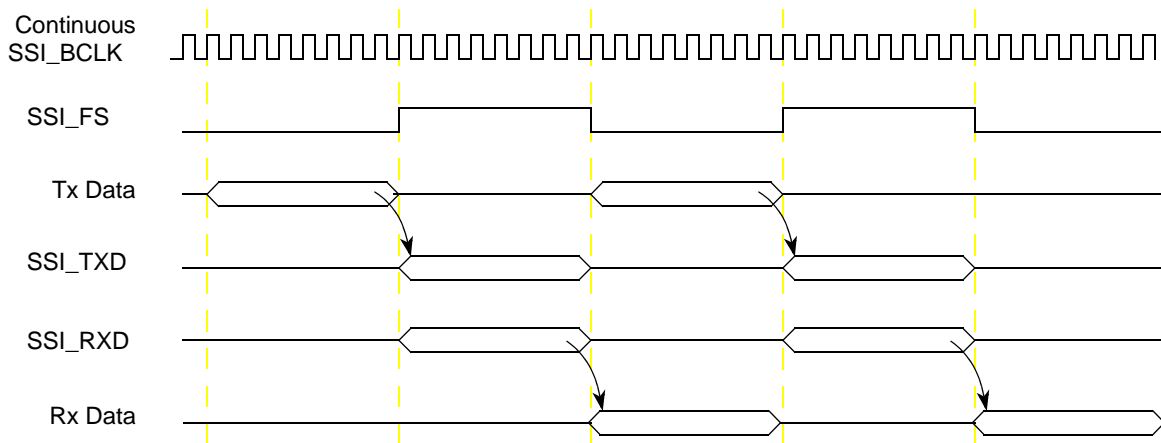
Figure 24-27. Normal Mode Timing - Continuous Clock

Figure 24-28 shows a similar case for internal (SSI generates clock) gated clock mode, and Figure 24-29 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).
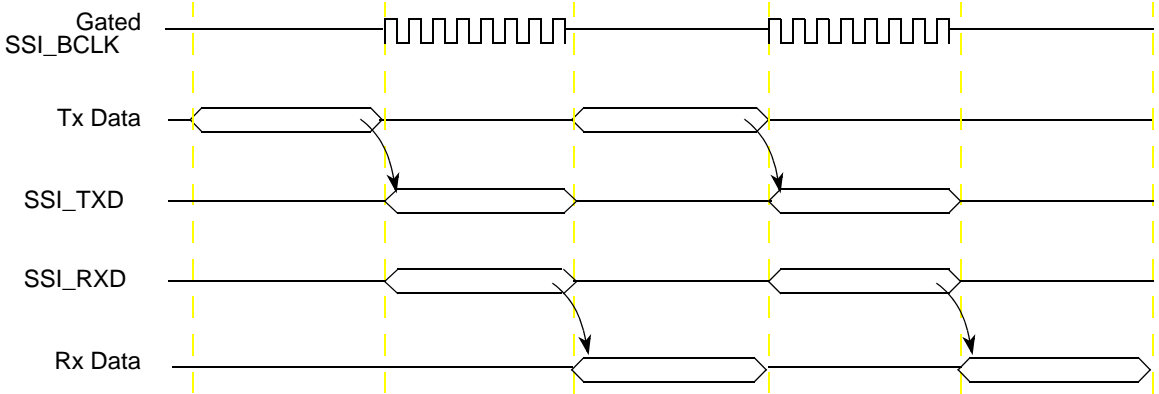
Figure 24-28. Normal Mode Timing - Internal Gated Clock
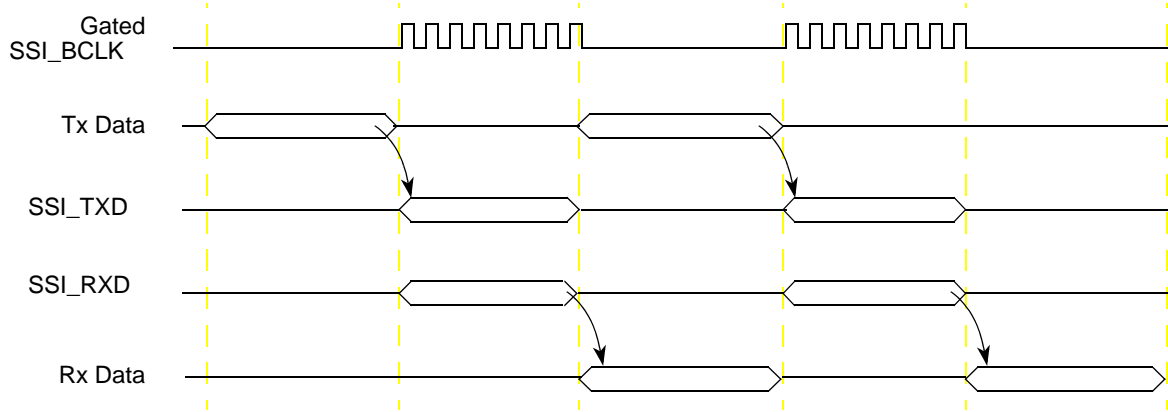
Figure 24-29. Normal Mode Timing - External Gated Clock

24.4.1.2   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 24.3.18, "SSI Transmit Time Slot Mask Register (SSI_TMASK)," and Section 24.3.19, "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).

24.4.1.2.1 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).

24.4.1.2.2 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 24-30 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.
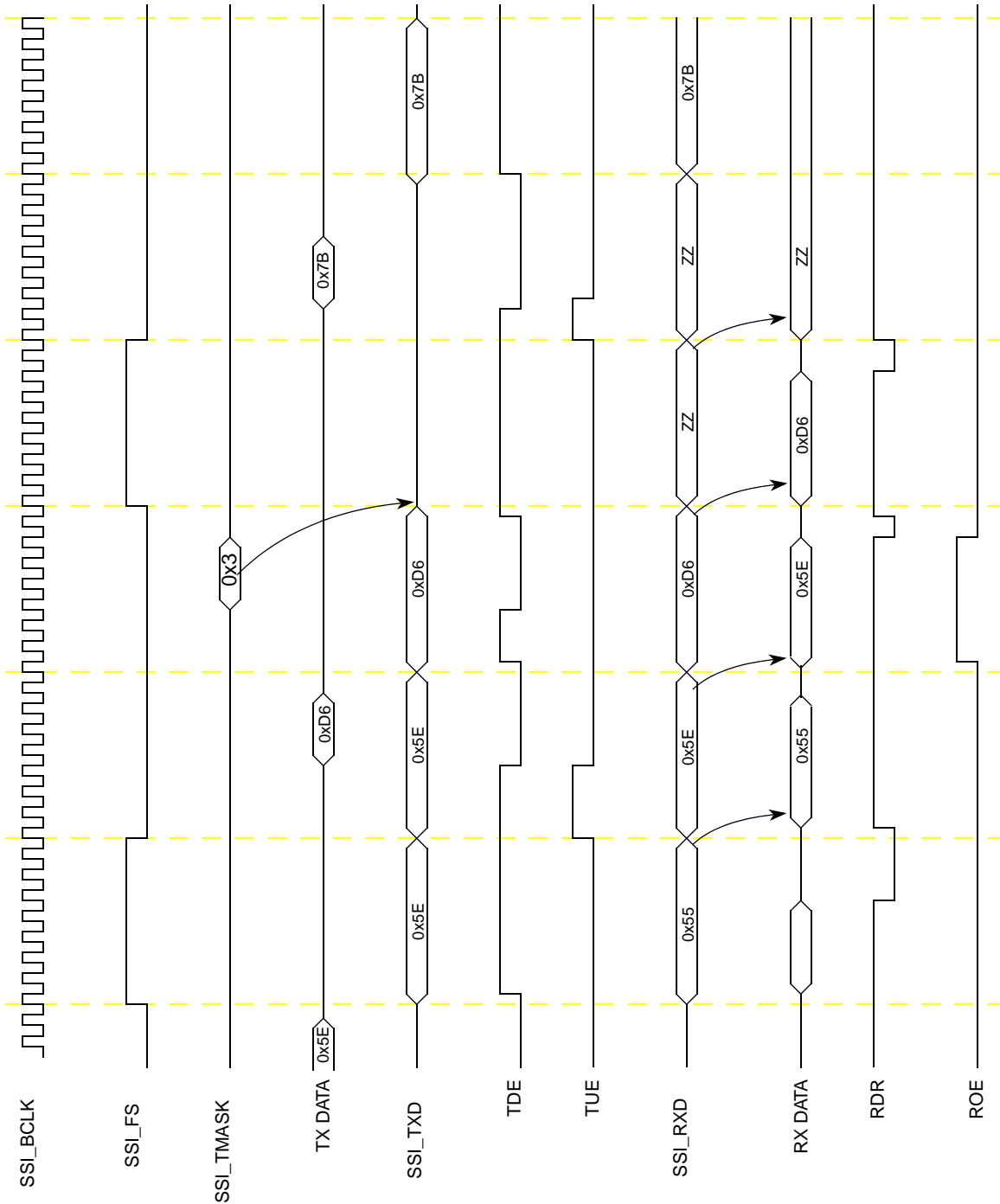
Figure 24-30. Network Mode Timing - Continuous Clock

24.4.1.3   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 24-3 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.

TSCKP=0, RSCKP=0
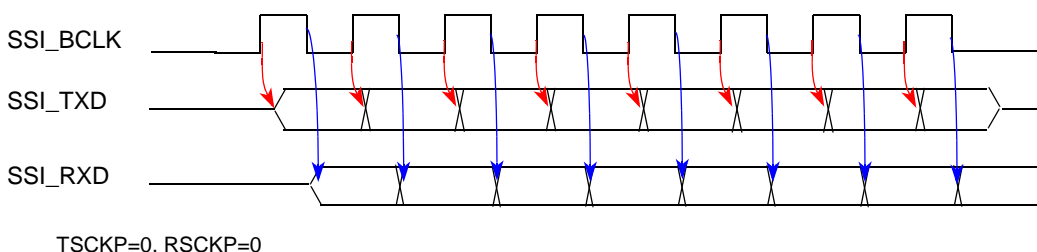
Figure 24-31. Internal Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching

TSCKP=1, RSCKP=1
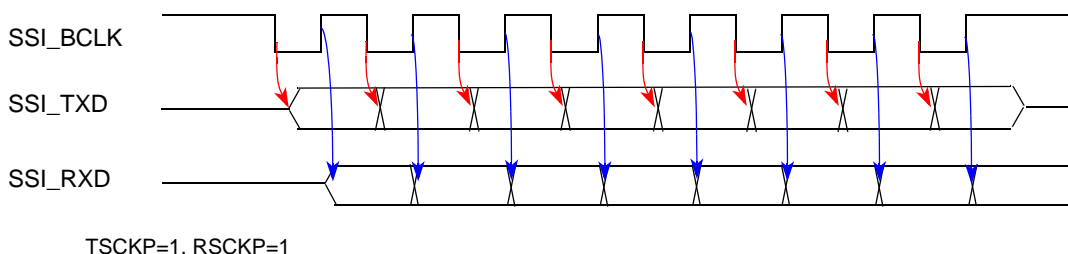
Figure 24-32. Internal Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

TSCKP=0, RSCKP=0
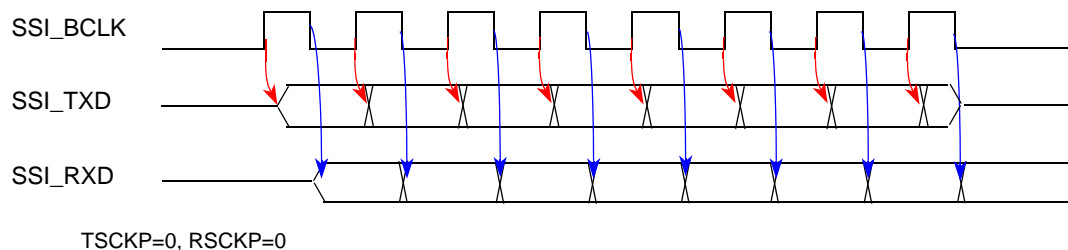
Figure 24-33. External Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching

TSCKP=1, RSCKP=1
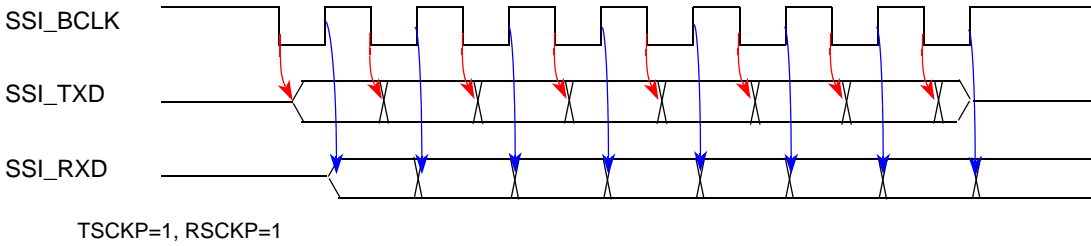
Figure 24-34. External Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

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.

24.4.1.4 I^{2}S Mode

The SSI is compliant to I^{2}S bus specification from Philips Semiconductors (February 1986, Revised June 5, 1996). Figure 24-35 depicts basic I^{2}S protocol timing.
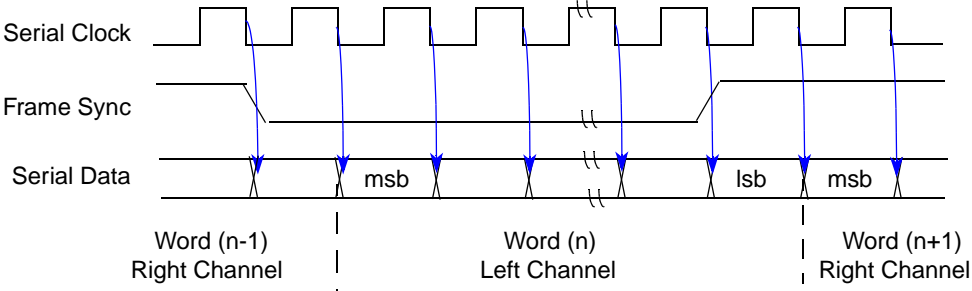
Figure 24-35. I^{2}S Mode Timing - Serial Clock, Frame Sync and Serial Data

I^{2}S mode can be selected by the SSI_CR[I2S] bits as follows:

Table 24-21. I^{2}S Mode Selection

SSI_CR[I2S]	Mode
00	Normal mode
01	I^{2}S master mode
10	I^{2}S slave mode
11	Normal mode

In normal (non-I^{2}S) 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^2S 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)

24.4.1.4.1 I^2S Master Mode

In I^2S 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^2S 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^2S 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.

24.4.1.4.2 I^2S Slave Mode

In I^2S 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.

24.4.1.5 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.

24.4.1.5.1 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) vaules are stored in the corresponding registers. The received data (slot 3–12) is then stored in the receive FIFO (for valid slots).

24.4.1.5.2 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.

24.4.2 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 (180 or 240 MHz) 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 ($f_{sys/3}$).

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^2S mode, the oversampling clock is available on this port if the SSI_CR[MCE] bit is set.

Figure 24-36 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 24-37.
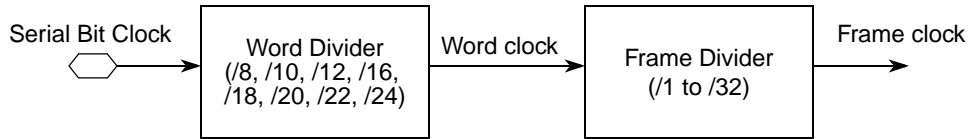
Figure 24-36. SSI Clocking

24.4.2.1   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 24-37 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.
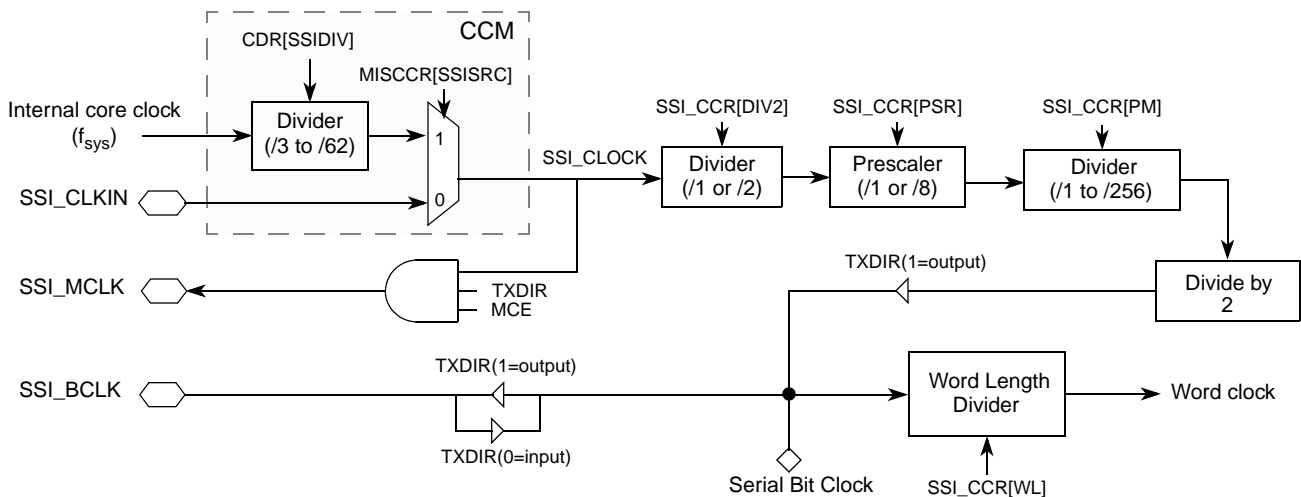
Figure 24-37. SSI Transmit Clock Generator Block Diagram

Figure 24-38 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.
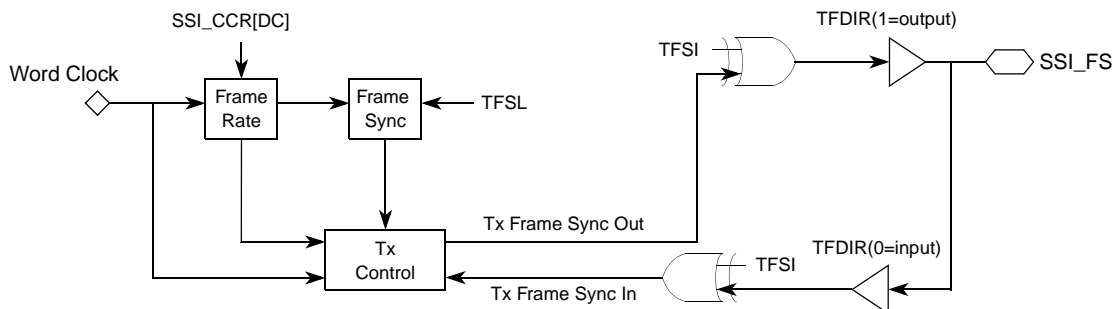
Figure 24-38. SSI Transmit Frame Sync Generator Block Diagram

24.4.2.2 DIV2, PSR and PM Bit Description

The bit clock frequency can be calculated from the SSI serial system clock (SSI_CLOCK), using Equation 24-1.

NOTE

> You must ensure that the bit-clock frequency is at most one-fifth the internal bus frequency ($f_{sys/3}$). The oversampling clock frequency can go up to internal bus frequency. Bits DIV2, PSR, and PM must not be cleared at the same time.

$$f_{INT\_BIT\_CLK} = \frac{\text{SSI serial system clock}}{(DIV2 + 1) \times (7 \times PSR + 1) \times (PM + 1) \times 2} \qquad \textit{Eqn. 24-1}$$

From this, the frame clock frequency can be calculated:

$$f_{FS\_CLK} = \frac{f_{INT\_BIT\_CLK}}{(DC + 1) \times (2 \times (WL + 1))} \qquad \textit{Eqn. 24-2}$$

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 24-22 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 180 or 240 MHz frequencies, the SSI module is not able to accurately produce standard bit and sample rates.

Table 24-22. 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

MCF5329 Reference Manual, Rev 3

Table 24-22. 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 24-23 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 24-23. 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

24.4.3 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.

24.4.4 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 24-24 shows the bit assignment for all the data formats supported by the SSI module.

Table 24-24. 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 24.3.10, "SSI Transmit Configuration Register (SSI_TCR)," and Section 24.3.11, "SSI Receive Configuration Register (SSI_RCR)," for more detail on the relevant bits in the SSI_TCR and SSI_RCR registers.

24.4.5 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 24-25 shows the conditions these interrupts are generated.

Table 24-25. SSI Receive Data Interrupts

Interrupt	RIE	ROEn	RFFn/RDRn
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

24.4.6 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 24-26 shows the conditions under which these interrupts are generated.

Table 24-26. 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

24.5 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 24-4
- 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 24-27).
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 24-27.

NOTE

These control bits should not be changed when the SSI module is enabled.

Table 24-27. 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 25
Real-Time Clock

25.1 Introduction

Figure 25-1 is a block diagram of the functional organization of the real time clock (RTC) module, consisting of:

- Prescaler
- Time-of-day (TOD) clock counter
- Alarm
- Sampling timer
- Minute stopwatch
- Associated control and bus interface hardware
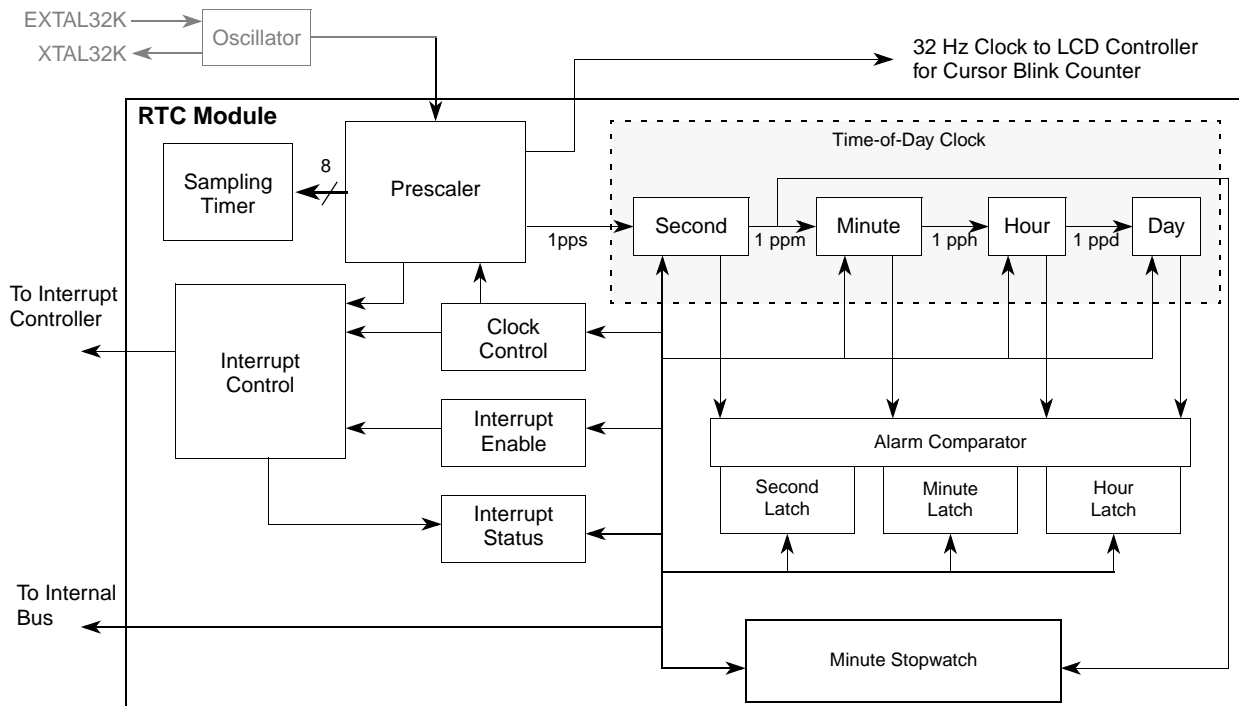
Figure 25-1. Real Time Clock Block Diagram

25.1.1 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.

25.1.2 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 at 32.768 kHz, 32 kHz, or 38.4 kHz (determined by reference clock crystal)

25.1.3 Modes of Operation

The real-time-clock operates in various modes as described below:

- Time-of-day counters
  - The prescaler divides the reference clock down to 1 Hz. The reference frequencies of 32.768 kHz, 38.4 kHz and 32 kHz are supported.
  - 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 exact time that the alarm should generate an interrupt. When the TOD clock value and the alarm value coincide, an interrupt occurs.
- Sampling Timer
  - The prescaler divides the reference clock down to 512 Hz. 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 25-13 lists the interrupt frequencies of the sampling timer for the possible reference clocks.
- Minute Stopwatch
  - The minute stopwatch performs a countdown with a one minute resolution. It generates an interrupt on a minute boundary.

25.2 External Signal Description

The below table describes the RTC external signals.

Table 25-1. RTC Signals

Signal Name	Abbreviation	Function	I/O
32 kHz External Clock In	EXTAL32K	32 kHz, 32.768 kHz, or 38.4 kHz crystal input clock.	I
32 kHz Crystal	XTAL32K	Oscillator output to crystal.	O

25.3 Memory Map/Register Definition

The RTC module includes ten registers, which are summarized below.

Table 25-2. Real Time Clock Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_8000	RTC Hours and Minutes Counter Register (RTC_HOURMIN)	32	R/W	Undefined	25.3.1/25-3
0xFC0A_8004	RTC Seconds Counter Register (RTC_SECONDS)	32	R/W	Undefined	25.3.2/25-4
0xFC0A_8008	RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)	32	R/W	0x0000_0000	25.3.3/25-4
0xFC0A_800C	RTC Seconds Alarm Register (RTC_ALRM_SEC)	32	R/W	0x0000_0000	25.3.4/25-5
0xFC0A_8010	RTC Control Register (RTC_CR)	32	R/W	0x0000_0080	25.3.5/25-5
0xFC0A_8014	RTC Interrupt Status Register (RTC_ISR)	32	R/W	0x0000_0000	25.3.6/25-6
0xFC0A_8018	RTC Interrupt Enable Register (RTC_IER)	32	R/W	0x0000_0000	25.3.7/25-7
0xFC0A_801C	Stopwatch Minutes Register (RTC_STPWCH)	32	R/W	0x0000_003F	25.3.8/25-8
0xFC0A_8020	RTC Days Counter Register (RTC_DAYS)	32	R/W	0x0000_0000	25.3.9/25-9
0xFC0A_8024	RTC Days Alarm Register (RTC_ALRM_DAY)	32	R/W	0x0000_0000	25.3.10/25-9

25.3.1 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: 0xFC0A_8000 (RTC_HOURMIN)   Access: User read/write
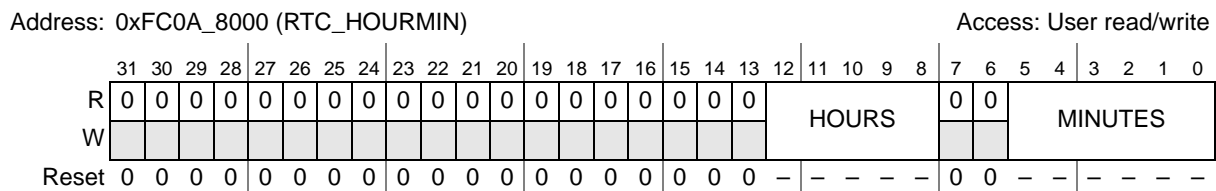
Figure 25-2. RTC Hours and Minutes Counter Register (RTC_HOURMIN)

Table 25-3. 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).

25.3.2 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: 0xFC0A_8004 (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	–	–	–	–	–	–
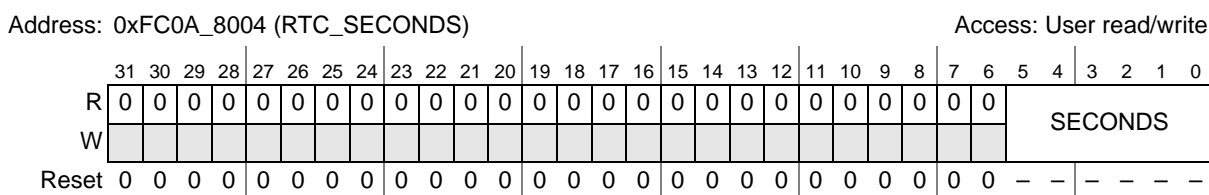
Figure 25-3. RTC Seconds Counter Register (RTC_SECONDS)

Table 25-4. 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).

25.3.3 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: 0xFC0A_8008 (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
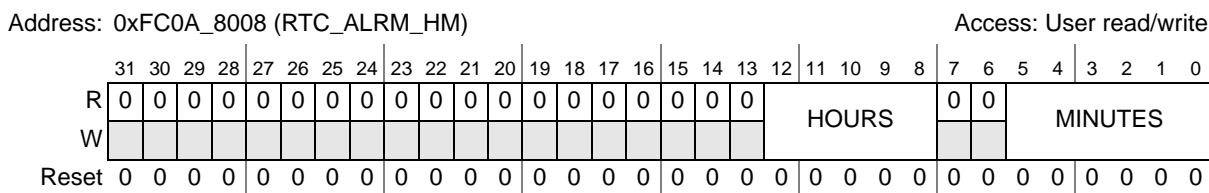
Figure 25-4. RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)

Table 25-5. 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).

25.3.4 RTC Seconds Alarm Register (RTC_ALRM_SEC)

The RTC_ALRM_SEC register configures the seconds setting for the alarm. The alarm settings can be read or written at any time.

Address: 0xFC0A_800C (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
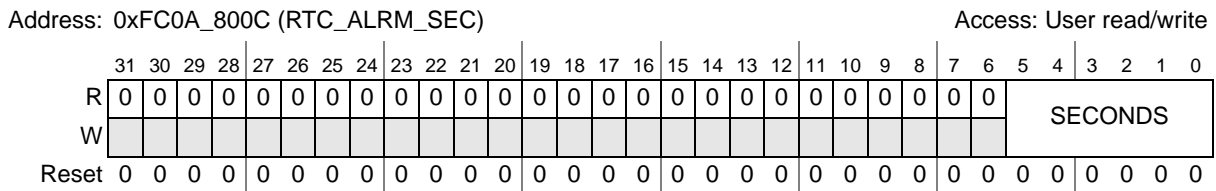
Figure 25-5. RTC Seconds Alarm Register (RTC_ALRM_SEC)

Table 25-6. RTC_ALRM_SEC Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5–0 SECONDS	Seconds setting of the alarm. Set to any value between 0 and 59 (0x3B).

25.3.5 RTC Control Register (RTC_CR)

The RTC_CR register enables the real-time clock module and software reset, and specifies the reference frequency information for the prescaler.

Address: 0xFC0A_8010 (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	\multicolumn{2}{c}{XTL}	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
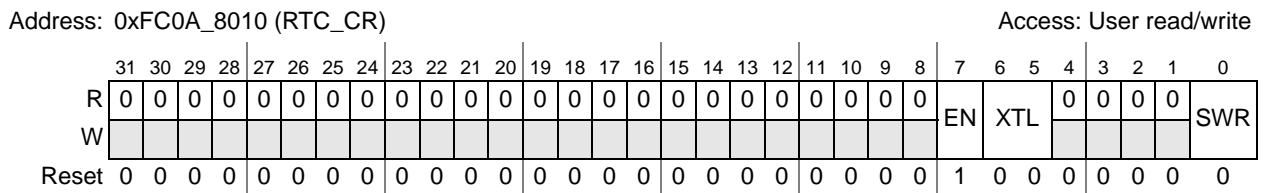
Figure 25-6. RTC Control Register (RTC_CR)

Table 25-7. 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
6–5 XTL	Crystal selection. Selects the proper input crystal frequency. It is important to set these bits appropriately or the RTC is inaccurate. 00 Input crystal frequency is 32.768 kHz 01 Input crystal frequency is 32 kHz 10 Input crystal frequency is 38.4 kHz 11 Input crystal frequency is 32.768 kHz
4–1	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

25.3.6 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: 0xFC0A_8014 (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 25-7. RTC Interrupt Status Register (RTC_ISR)

Table 25-8. RTC_ISR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 SAMn	Sampling timer 7–0 interrupt flags. Indicates an interrupt has occurred at the corresponding sampling rate. See Section 25.4.3, "Sampling Timer," for more details. 0 No SAM7–0 interrupt has occurred 1 A SAM7–0 interrupt has occurred

Table 25-8. RTC_ISR Field Descriptions (continued)

Field	Description
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. 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

25.3.7 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: 0xFC0A_8018 (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 W	SAM7	SAM6	SAM5	SAM4	SAM3	SAM2	SAM1	SAM0	2HZ	0	HR	1HZ	DAY	ALM	MIN	SW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-8. RTC Interrupt Enable Register (RTC_IER)

Table 25-9. 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.

25.3.8 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: 0xFC0A_801C (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
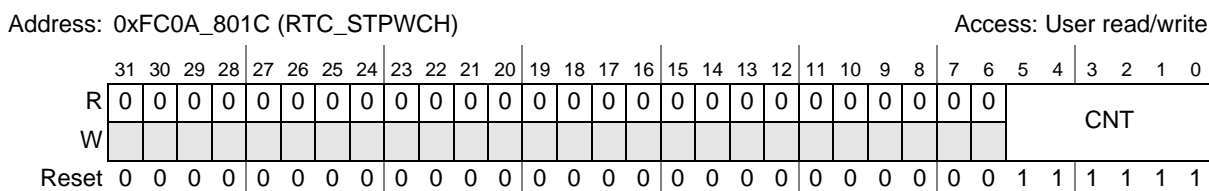
Figure 25-9. RTC Stopwatch Minutes Register (RTC_STPWCH)

Table 25-10. 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.

25.3.9 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: 0xFC0A_8020 (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
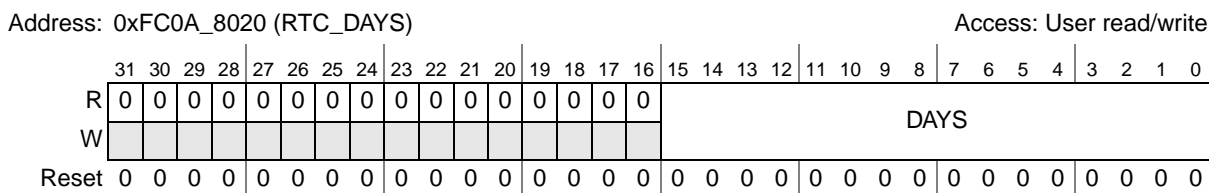
Figure 25-10. RTC Days Counter Register (RTC_DAYS)

Table 25-11. 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).

25.3.10 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: 0xFC0A_8024 (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
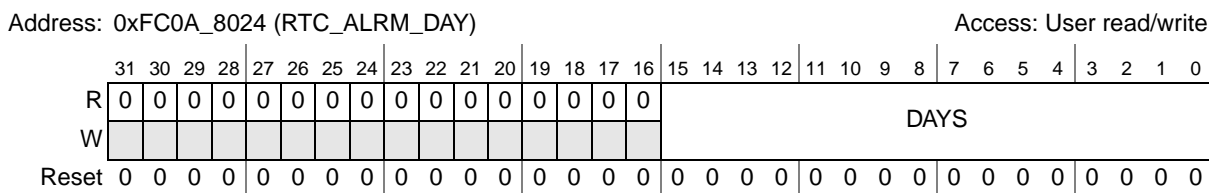
Figure 25-11. RTC Day Alarm Register (RTC_ALRM_DAY)

Table 25-12. 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).

25.4 Functional Description

The prescaler converts the incoming crystal reference clock to 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.

25.4.1 Prescaler and Counter

The prescaler divides the reference clock down to 1 Hz. The reference frequencies of 32.768 kHz, 38.4 kHz and 32 kHz are supported, selected via the RTC_CR[XTL] bit field. The prescaler stages are tapped to support the sampling timer.

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.

25.4.2 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 exact timethat 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. 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 25.5, "Initialization/Application Information," for the correct procedure to follow when changing the alarm or time-of-day (day, hour, minute, or second) registers.

25.4.3 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 following table lists the interrupt frequencies of the sampling timer for the possible reference clocks.

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 25-13. Sampling Timer Frequencies

Sampling Frequency	32.768 kHz Reference Clock	32 kHz Reference Clock	38.4 kHz Reference Clock
SAM7	512 Hz	500 Hz	600 Hz
SAM6	256 Hz	250 Hz	300 Hz
SAM5	128 Hz	125 Hz	150 Hz
SAM4	64 Hz	62.50 Hz	75 Hz
SAM3	32Hz	31.25 Hz	37.50 Hz
SAM2	16 Hz	15.625 Hz	18.75 Hz
SAM1	8 Hz	7.81 Hz	9.38 Hz
SAM0	4 Hz	3.91 Hz	4.69 Hz
2HZ	2 Hz	1.96 Hz	2.35 Hz

25.4.4 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 the LCD controller 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.

25.5 Initialization/Application Information

25.5.1 Flow Chart of RTC Operation

Table 25-12 shows the flow chart of a typical RTC operation.
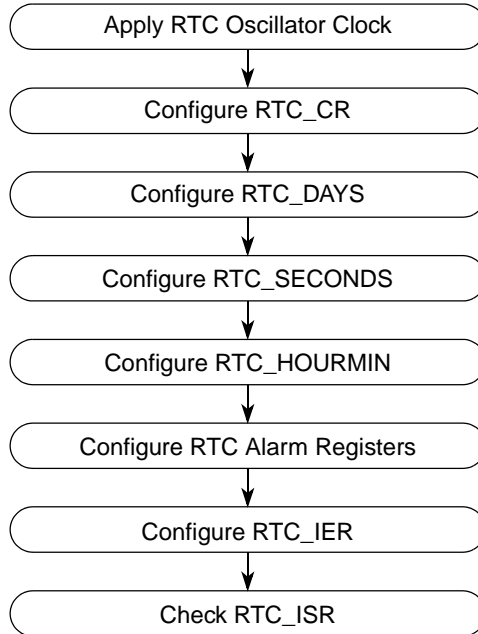
Figure 25-12. Flow Chart of RTC Operation

25.5.2 Programming the Alarm or Time-of-Day Registers

Use the following procedure illustrated in Figure 25-13 when changing the alarm or time-of-day (day, hour, minute, and second) registers.
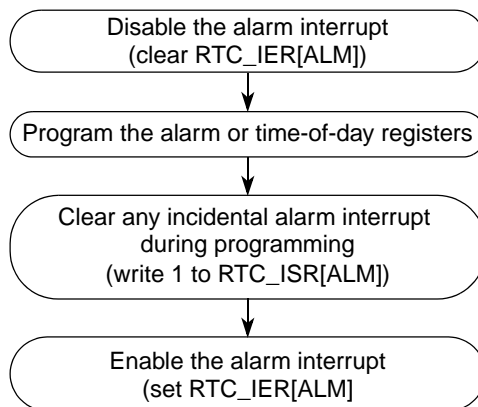
Figure 25-13. Flow Chart of Alarm and Time-of-Day Programming

Chapter 26
Pulse-Width Modulation (PWM) Module

26.1 Introduction

This chapter describes the configuration and operation of the pulse-width modulation (PWM) module. It includes a block diagram, programming model, and functional description.

26.1.1 Overview

The PWM module, shown in Figure 26-1, generates a synchronous series of pulses having programmable period and duty cycle. With a suitable low-pass filter, the PWM can be used as a digital-to-analog converter.
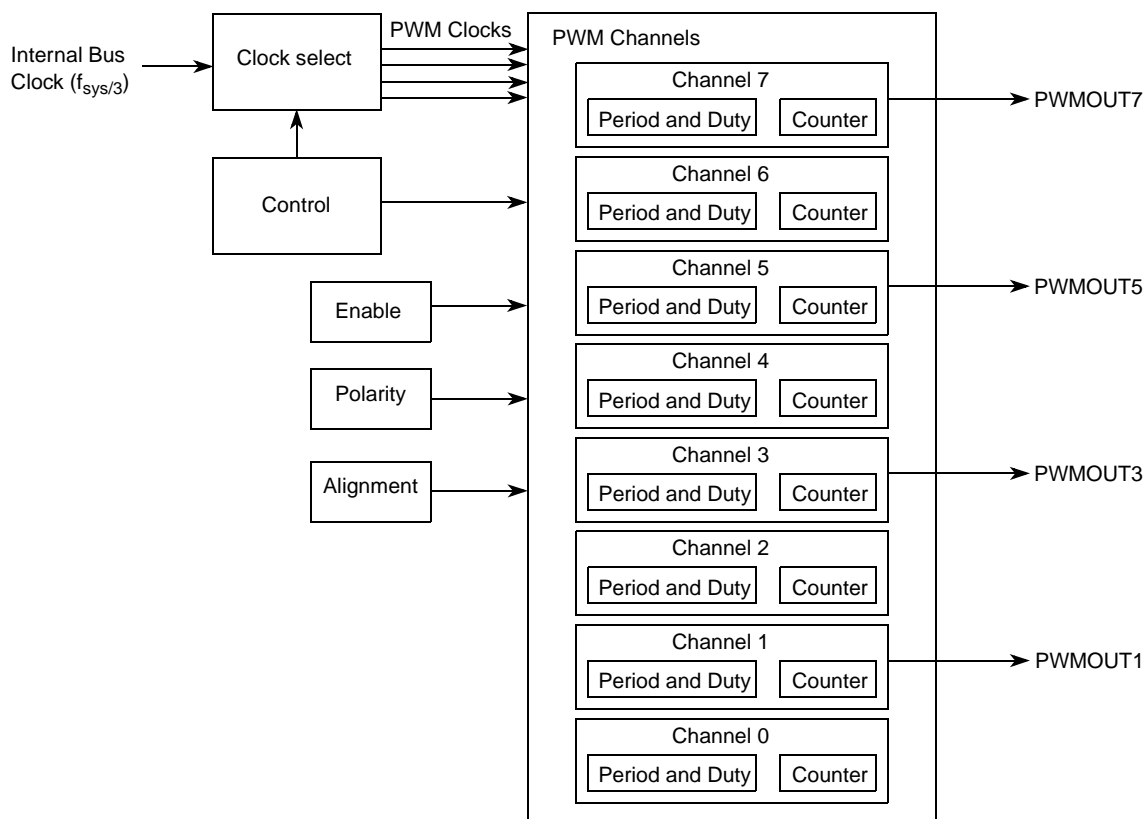
Figure 26-1. PWM Block Diagram

Main features include the following:

- Double-buffered period and duty cycle
- Left- or center-aligned outputs
- Eight independent PWM modules. Notice that only the four odd PWM channel outputs are available on the device. The even channels can be used for concatenation purposes to generate 16-bit PWM for the odd channels.
- Byte-wide registers provide programmable duty cycle and period control
- Four programmable clock sources

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 13, "General Purpose I/O Module"") prior to configuring the PWM module.

26.2 Memory Map/Register Definition

This section describes the registers and control bits in the PWM module. There are eight independent PWM modules, each with its own control and counter registers, although only four channels have an output signal. The memory map for the PWM is shown below.

NOTE

Longword accesses to any of the PWM registers result in a bus error. Only byte and word accesses are allowed.

Table 26-1. PWM Memory Map

Address[1,2]	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC09_0020	PWM Enable Register (PWME)	8	R/W	0x00	26.2.1/26-3
0xFC09_0021	PWM Polarity Register (PWMPOL)	8	R/W	0x00	26.2.2/26-4
0xFC09_0022	PWM Clock Select Register (PWMCLK)	8	R/W	0x00	26.2.3/26-4
0xFC09_0023	PWM Prescale Clock Select Register (PWMPRCLK)	8	R/W	0x00	26.2.4/26-5
0xFC09_0024	PWM Center Align Enable Register (PWMCAE)	8	R/W	0x00	26.2.5/26-6
0xFC09_0025	PWM Control Register (PWMCTL)	8	R/W	0x00	26.2.6/26-6
0xFC09_0028	PWM Scale A Register (PWMSCLA)	8	R/W	0x00	26.2.7/26-7
0xFC09_0029	PWM Scale B Register (PWMSCLB)	8	R/W	0x00	26.2.8/26-8
0xFC09_002C + n $n = 0–7$	PWM Channel n Counter Register (PWMCNTn)	8	R/W	0x00	26.2.9/26-9
0xFC09_0034 + n $n = 0–7$	PWM Channel n Period Register (PWMPERn)	8	R/W	0xFF	26.2.10/26-10
0xFC09_003C + n $n = 0–7$	PWM Channel n Duty Register (PWMDTYn)	8	R/W	0xFF	26.2.11/26-10
0xFC09_0044	PWM Shutdown Register (PWMSDN)	8	R/W	0x00	26.2.12/26-11

MCF5329 Reference Manual, Rev 3

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

[2] A 32-bit access to any of these registers results in a bus transfer error (see Section 11.2.7, "SCM Interrupt Status Register (SCMISR)").

26.2.1 PWM Enable Register (PWME)

Each PWM channel has an enable bit (PWME*n*) to start its waveform output. While in run mode, if all four PWM output channels are disabled (PWME[7:0] = 0), the prescaler counter shuts off for power savings. See Section 26.3.2.1, "PWM Enable" for more information.

Address: 0xFC09_0020 (PWME)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R	PWME7	0	PWME5	0	PWME3	0	PWME1	0
W								
Reset:	0	0	0	0	0	0	0	0
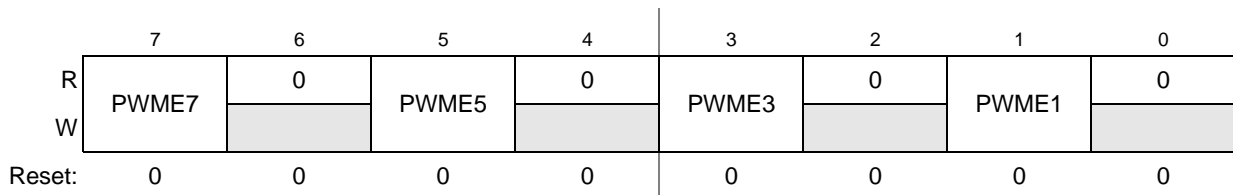
Figure 26-2. PWM Enable Register (PWME)

Table 26-2. PWME Field Descriptions

Field	Description
7 PWME7	PWM Channel 7 Enable. In normal mode, if enabled, the PWM signal becomes available at PWMOUT7 when its corresponding clock source begins its next cycle. When PWMSDN[SDNEN] is set this channel is an input for emergency shutdown. 0 PWM7 disabled 1 PWM7 enabled
6	Reserved, must be cleared.
5 PWME5	PWM Channel 5 Output Enable. If enabled, the PWM signal becomes available at PWMOUT5 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
4	Reserved, must be cleared.
3 PWME3	PWM Channel 3 Output Enable. If enabled, the PWM signal becomes available at PWMOUT3 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
2	Reserved, must be cleared.
1 PWME1	PWM Channel 1 Output Enable. If enabled, the PWM signal becomes available at PWMOUT1 when its corresponding clock source begins its next cycle. 0 PWM output disabled 1 PWM output enabled
0	Reserved, must be cleared.

26.2.2 PWM Polarity Register (PWMPOL)

The starting polarity of each PWM channel waveform is determined by the associated PWMPOL[PPOL*n*] bit. If the polarity is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

Address: 0xFC09_0021 (PWMPOL)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R	PPOL7	0	PPOL5	0	PPOL3	0	PPOL1	0
W								
Reset:	0	0	0	0	0	0	0	0
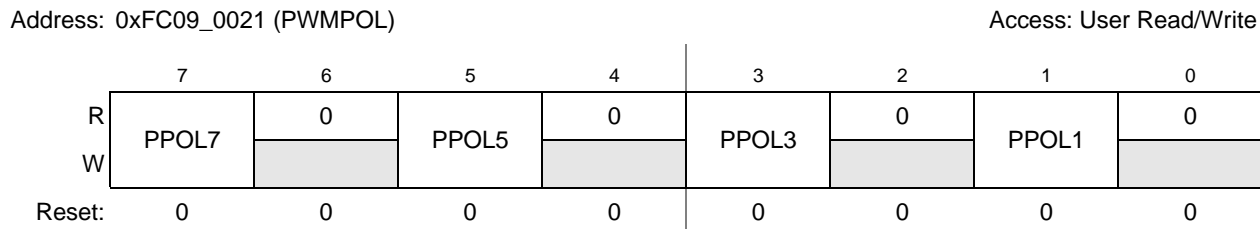
Figure 26-3. PWM Polarity Register (PWMPOL)

Table 26-3. PWMPOL Field Descriptions

Field	Description
7,5,3,1 PPOL*n*	PWM channel *n* polarity. 0 PWM channel *n* output is low at the beginning of the period, then goes high when the duty count is reached 1 PWM channel *n* output is high at the beginning of the period, then goes low when the duty count is reached
6,4,2,0	Reserved, must be cleared.

26.2.3 PWM Clock Select Register (PWMCLK)

Each PWM channel has the capability of selecting one of two clocks. For channels 1 and 5, the clock choices are clock A or SA. For channels 3 and 7, the choices are clock B or SB. The clock selection is done with the below PWMCLK[PCLK*n*] control bits. If a clock select is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

Address: 0xFC09_0022 (PWMCLK)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R	PCLK7	0	PCLK5	0	PCLK3	0	PCLK1	0
W								
Reset:	0	0	0	0	0	0	0	0
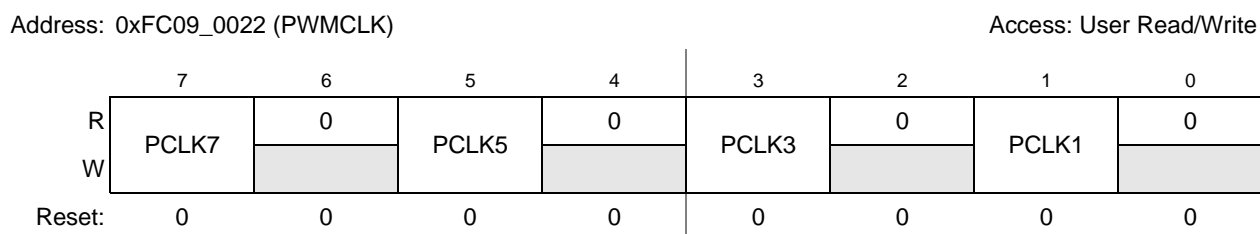
Figure 26-4. PWM Clock Select Register (PWMCLK)

Table 26-4. PWMCLK Field Descriptions

Field	Description
7,5,3,1 PCLK*n*	PWM channel *n* clock select. Selects between one of two clock sources for each PWM channel. See Section 26.2.4, "PWM Prescale Clock Select Register (PWMPRCLK)" and Section 26.2.7, "PWM Scale A Register (PWMSCLA)" for more information on how the different clock rates are generated. The even-numbered channels' clock select has no effect when the corresponding PWMCTL[CON*n(n+1)*] bit is set. For example, if PWMCTL[CON01] equals 1, PWMCLK[PCLK0] has no affect.

	PCLK7 (PCLK7 Clock Source)	PCLK5 (PWM5 Clock Source)	PCLK3 (PWM3 Clock Source)	PCLK1 (PWM1 Clock Source)
0	B	A	B	A
1	SB	SA	SB	SA

Field	Description
6,4,2,0	Reserved, must be cleared.

26.2.4 PWM Prescale Clock Select Register (PWMPRCLK)

The PWMPRCLK register selects the prescale clock source for clocks A and B independently. If the clock prescale is changed while a PWM signal is being generated, a truncated or stretched pulse can occur during the transition.

Address: 0xFC09_0023 (PWMPRCLK)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R	0		PCKB		0		PCKA	
W								
Reset:	0	0	0	0	0	0	0	0
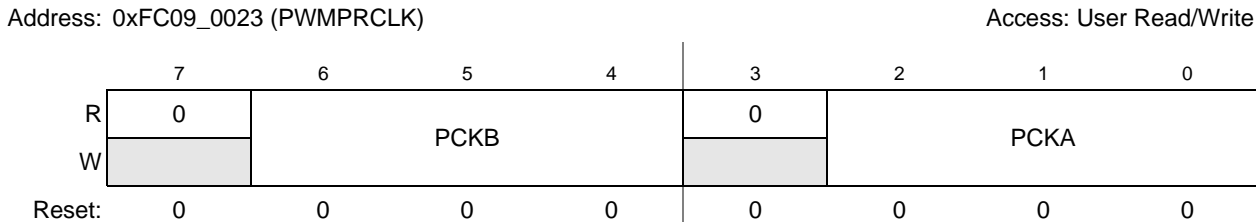
Figure 26-5. PWM Prescale Clock Select Register (PWMPRCLK)

Table 26-5. PWMPRCLK Field Descriptions

Field	Description
7	Reserved, must be cleared.
6–4 PCKB	Clock B prescaler select. These three bits control the rate of Clock B, which can be used for PWM channels 3 and 7.

PCKB	Clock B Rate
000	Internal bus clock $\div 2^0$
001	Internal bus clock $\div 2^1$
...	...
111	Internal bus clock $\div 2^7$

Table 26-5. PWMPRCLK Field Descriptions (continued)

Field	Description
3	Reserved, must be cleared.
2–0 PCKA	Clock A prescaler select. These three bits control the rate of Clock A, which can be used for PWM channels 1 and 5.

PCKA	Clock A Rate
000	Internal bus clock $\div 2^0$
001	Internal bus clock $\div 2^1$
...	...
111	Internal bus clock $\div 2^7$

26.2.5 PWM Center Align Enable Register (PWMCAE)

The PWMCAE register contains four control bits for the selection of center-aligned outputs or left-aligned outputs for each PWM channel. Write these bits only when the corresponding channel is disabled. See Section 26.3.2.5, "Left-Aligned Outputs" and Section 26.3.2.6, "Center-Aligned Outputs" for a more detailed description of the PWM output modes.

Address: 0xFC09_0024 (PWMCAE)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R	CAE7	0	CAE5	0	CAE3	0	CAE1	0
W								
Reset:	0	0	0	0	0	0	0	0
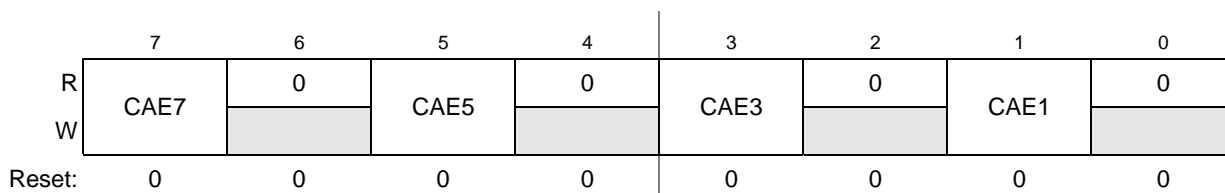
Figure 26-6. PWM Center Align Enable Register (PWMCAE)

Table 26-6. PWMCAE Field Descriptions

Field	Description
7,5,3,1 CAE*n*	Center align enable for channel *n*. The even-numbered channels' center align enable has no effect when the corresponding PWMCTL[CON*n(n+1)*] bit is set. For example, if PWMCTL[CON01] equals 1, PWMCAE[CAE0] has no affect. 0 Channel *n* operates in left-aligned output mode 1 Channel *n* operates in center-aligned output mode
6,4,2,0	Reserved, must be cleared.

26.2.6 PWM Control Register (PWMCTL)

The PWMCTL register provides various control of the PWM module. Change the CON*n(n+1)* bits only when both corresponding channels are disabled. See Section 26.3.2.7, "PWM 16-Bit Functions" for a more detailed description of the concatenation function.

Address: 0xFC09_0025 (PWMCTL) Access: User Read/Write

	7	6	5	4	3	2	1	0
R	CON67	CON45	CON23	CON01	PSWAI	PFRZ	0	0
W								
Reset:	0	0	0	0	0	0	0	0
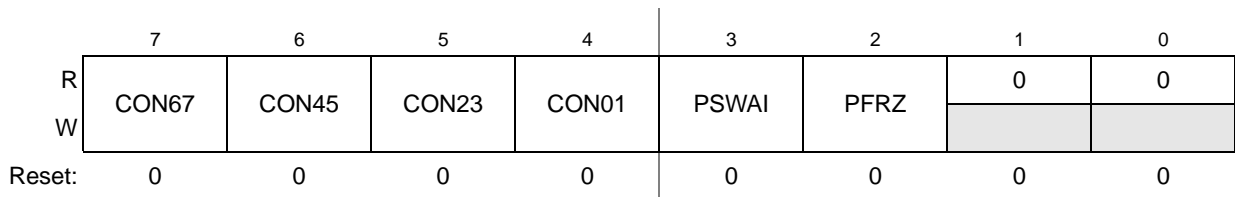
Figure 26-7. PWM Control Register (PWMCTL)

Table 26-7. PWMCTL Field Descriptions

Field	Description
7 CON67	Concatenates PWM channels 6 and 7 to form one 16-bit PWM channel. 0 Channels 6 and 7 are separate 8-bit PWMs. There is no PWM 6 output. 1 Concatenate PWM 6 and 7. Channel 6 becomes the high order byte and channel 6 the low order byte. PWMOUT7 is the output for this 16-bit PWM signal, and PWMOUT6 is disabled. The channel 7 clock select, polarity, center align enable, and enable bits control this concatenated output.
6 CON45	Concatenates PWM channels 4 and 5 to form one 16-bit PWM channel. 0 Channels 4 and 5 are separate 8-bit PWMs. There is no PWM 4 output. 1 Concatenate PWM 4 and 5. Channel 4 becomes the high order byte and channel 5 the low order byte. PWMOUT5 is the output for this 16-bit PWM signal, and PWMOUT4 is disabled. The channel 5 clock select, polarity, center align enable, and enable bits control this concatenated output.
5 CON23	Concatenates PWM channels 2 and 3 to form one 16-bit PWM channel. 0 Channels 2 and 3 are separate 8-bit PWMs. There is no PWM 2 output. 1 Concatenate PWM 2 and 3. Channel 2 becomes the high order byte and channel 3 the low order byte. PWMOUT3 is the output for this 16-bit PWM signal, and PWMOUT2 is disabled. The channel 3 clock select, polarity, center align enable, and enable bits control this concatenated output.
4 CON01	Concatenates PWM channels 0 and 1 to form one 16-bit PWM channel. 0 Channels 0 and 1 are separate 8-bit PWMs. There is no PWM 0 output. 1 Concatenate PWM 0 and 1. Channel 0 becomes the high order byte and channel 1 the low order byte. PWMOUT1 is the output for this 16-bit PWM signal, and PWMOUT0 is disabled. The channel 1 clock select, polarity, center align enable, and enable bits control this concatenated output.
3 PSWAI	PWM stops in doze mode. Disables the input clock to the prescaler while in doze mode. 0 Allow the clock to the prescaler while in doze mode 1 Stop the input clock to the prescaler when the core is in doze mode
2 PFRZ	PWM counters stop in debug mode ($\overline{\text{BKPT}}$ asserted). 0 Allow PWM counters to continue while in debug mode 1 Disable PWM input clock to the prescaler when the core is in debug mode. Useful for emulation as it allows the PWM function to be suspended.
1–0	Reserved, must be cleared.

26.2.7   PWM Scale A Register (PWMSCLA)

PWMSCLA is the programmable scale value used in scaling clock A to generate clock SA. Clock SA is generated with the following equation:

$$\text{Clock SA} \ = \ \frac{\text{Clock A}}{2 \times \text{PWMSCLA}}$$

Eqn. 26-1

Any value written to this register causes the scale counter to load the new scale value (PWMSCLA).

Address: 0xFC09_0028 (PWMSCLA)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R								
W				SCALEA				
Reset:	0	0	0	0	0	0	0	0
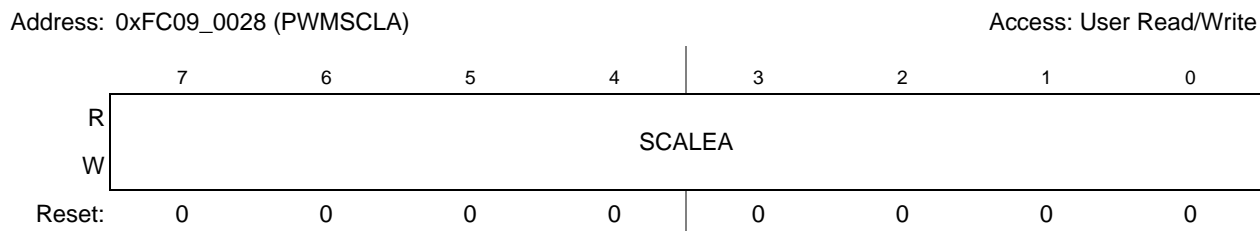
Figure 26-8. PWM Scale A Register (PWMSCLA)

Table 26-8. PWMSCLA Field Descriptions

Field	Description
7–0 SCALEA	Part of divisor used to form Clock SA from Clock A. <table><tr><th>SCALEA</th><th>Value</th></tr><tr><td>0x00</td><td>256</td></tr><tr><td>0x01</td><td>1</td></tr><tr><td>0x02</td><td>2</td></tr><tr><td>...</td><td>...</td></tr><tr><td>0xFF</td><td>255</td></tr></table>

26.2.8 PWM Scale B Register (PWMSCLB)

PWMSCLB is the programmable scale value used in scaling clock B to generate clock SB. Clock SB is generated according to the following equation:

$$\text{Clock SB} = \frac{\text{Clock B}}{2 \times \text{PWMSCLB}}$$

Eqn. 26-2

Any value written to this register causes the scale counter to load the new scale value (PWMSCLB).

Address: 0xFC09_0029 (PWMSCLB)   Access: User Read/Write

	7	6	5	4	3	2	1	0
R								
W				SCALEB				
Reset:	0	0	0	0	0	0	0	0
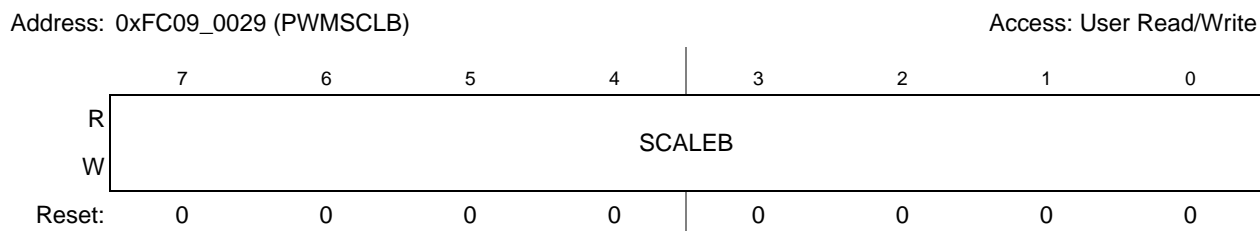
Figure 26-9. PWM Scale B Register (PWMSCLB)

Table 26-9. PWMSCLB Field Descriptions

Field	Description
7–0 SCALEB	Divisor used to form Clock SB from Clock B. | SCALEB	Value | |---|---| | 0x00 | 256 | | 0x01 | 1 | | 0x02 | 2 | | ... | ... | | 0xFF | 255 |

The table within the field description:

SCALEB	Value
0x00	256
0x01	1
0x02	2
...	...
0xFF	255

26.2.9 PWM Channel Counter Registers (PWMCNT*n*)

Each channel has a dedicated 8-bit up/down counter that runs at the rate of the selected clock source, PWMCLK[PCLK*n*]. The user can read the counters at any time without affecting the count or the operation of the PWM channel. In left-aligned output mode, the counter counts from 0 to the value in the period register minus 1. In center-aligned output mode, the counter counts from 0 up to the value in the period register and then back down to 0. Therefore, given the same value in the period register, center-aligned mode is twice the period of left-aligned mode.

Any value written to the counter causes the counter to reset to 0x00, the counter direction to be set to up for center-aligned mode, the immediate load of duty and period registers with values from the buffers, and the output to change according to the polarity bit.

The counter is also cleared at the end of the effective period (see Section 26.3.2.5, "Left-Aligned Outputs" and Section 26.3.2.6, "Center-Aligned Outputs" for more details). When the channel is disabled (PWME*n*=0), the PWMCNT*n* register does not count. When a channel is enabled (PWME*n*=1), the associated PWM counter starts at the count in the PWMCNT*n* register. For more detailed information on the operation of the counters, refer to Section 26.3.2.4, "PWM Timer Counters."

Address: 0xFC09_002C (PWMCNT0) Access: User Read/Write
0xFC09_002D (PWMCNT1)
0xFC09_002E (PWMCNT2)
0xFC09_002F (PWMCNT3)
0xFC09_0030 (PWMCNT4)
0xFC09_0031 (PWMCNT5)
0xFC09_0032 (PWMCNT6)
0xFC09_0033 (PWMCNT7)

	7	6	5	4	3	2	1	0
R				COUNT				
W								
Reset:	0	0	0	0	0	0	0	0
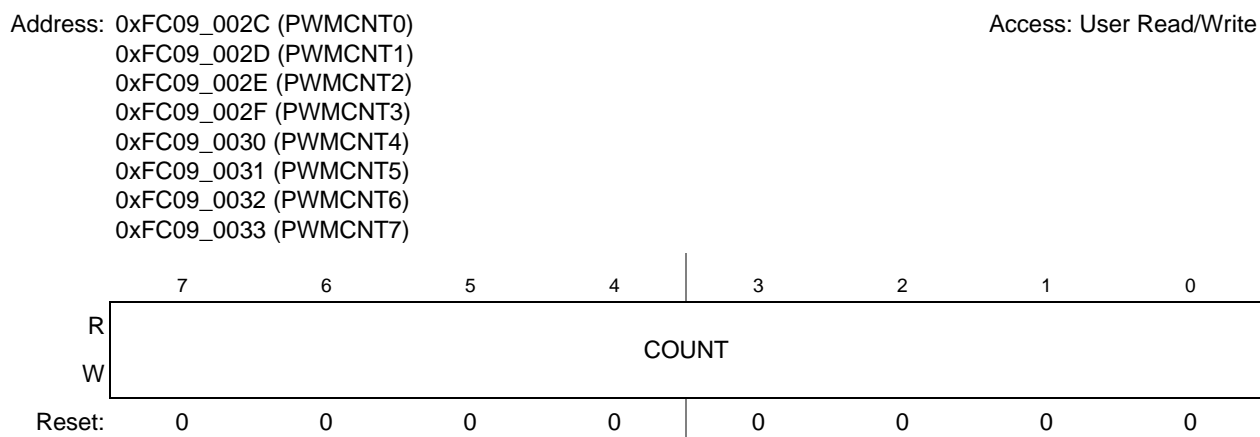
Figure 26-10. PWM Counter Registers (PWMCNT*n*)

MCF5329 Reference Manual, Rev 3

Table 26-10. PWMCNT*n* Field Descriptions

Field	Description
7–0 COUNT	Current value of the PWM up counter. Resets to zero when written.

26.2.10 PWM Channel Period Registers (PWMPER*n*)

The PWM period registers determine the period of the associated PWM channel. Refer to Section 26.3.2.3, "PWM Period and Duty" for more information.

Calculating the output period depends on the output mode (center-aligned has twice the period as left-aligned mode) as well as PWMPER*n*. See the below equation:

$$\text{PWM}n \text{ period} = \text{Channel clock period} \times (\text{PWMCAE}[\text{CAE}n] + 1) \times \text{PWMPER}n \qquad \textbf{Eqn. 26-3}$$

For boundary case programming values (e.g. PWMPER*n* = 0x00), please refer to Section 26.3.2.8, "PWM Boundary Cases".

Address: 0xFC09_0034 (PWMPER0) Access: User Read/Write
0xFC09_0035 (PWMPER1)
0xFC09_0036 (PWMPER2)
0xFC09_0037 (PWMPER3)
0xFC09_0038 (PWMPER4)
0xFC09_0039 (PWMPER5)
0xFC09_003A (PWMPER6)
0xFC09_003B (PWMPER7)

	7	6	5	4	3	2	1	0
R W				PERIOD				
Reset:	1	1	1	1	1	1	1	1
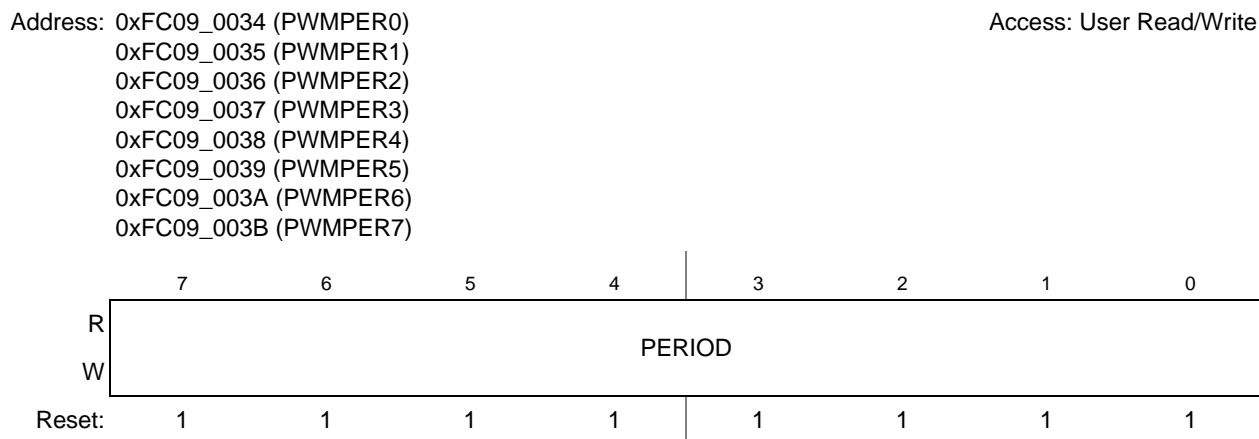
Figure 26-11. PWM Period Registers (PWMPER*n*)

Table 26-11. PWMPER*n* Field Descriptions

Field	Description
7–0 PERIOD	Period counter for the output PWM signal. If PERIOD equals 0x00, the PWM*n* output is always high (PPOL*n*=1) or always low (PPOL*n*=0). See Section 26.3.2.8, "PWM Boundary Cases" for other special cases.

26.2.11 PWM Channel Duty Registers (PWMDTY*n*)

The PWM duty registers determine the duty cycle of the associated PWM channel. To calculate the output duty cycle (high time as a percentage of period) for a particular channel:

$$\text{Duty Cycle} = \left| \left(1 - \text{PWMPOL}[\text{PPOL}n] - \frac{\text{PWMDTY}n}{\text{PWMPER}n} \right) \right| \times 100\% \qquad \textbf{Eqn. 26-4}$$

For boundary case programming values (e.g. PWMDTY*n* = 0x00 or PWMDTY*n* > PWMPER*n*), refer to Section Section 26.3.2.8, "PWM Boundary Cases".

Address: 0xFC09_003C (PWMDTY0)   Access: User Read/Write
            0xFC09_003D (PWMDTY1)
            0xFC09_003E (PWMDTY2)
            0xFC09_003F (PWMDTY3)
            0xFC09_0040 (PWMDTY4)
            0xFC09_0041 (PWMDTY5)
            0xFC09_0042 (PWMDTY6)
            0xFC09_0043 (PWMDTY7)

	7	6	5	4	3	2	1	0
R				DUTY				
W								
Reset:	1	1	1	1	1	1	1	1
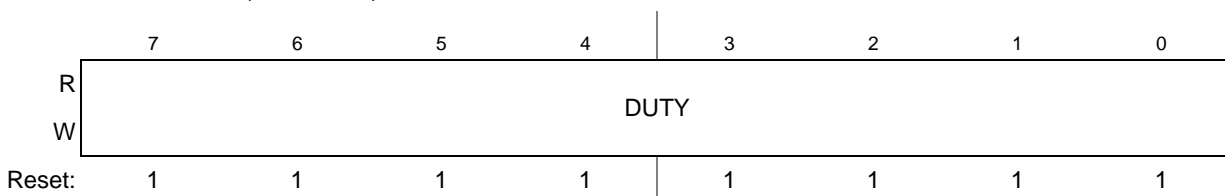
Figure 26-12. PWM Duty Registers (PWMDTY*n*)

Table 26-12. PWMDTY*n* Field Descriptions

Field	Description
7–0 DUTY	Contains the duty value used to determine when a transition occurs on the PWM output signal. When a match occurs with the corresponding PWMCNT*n* register, the PWM output toggles. If DUTY equals 0x00, the PWM*n* output is always low (PPOL*n*=1) or always high (PPOL*n*=0). See Section 26.3.2.8, "PWM Boundary Cases" for other special cases.

26.2.12 PWM Shutdown Register (PWMSDN)

The PWM shutdown register provides emergency shutdown functionality of the PWM module. The PWMSDN[7:1] bits are ignored if PWMSDN[SDNEN] is cleared.

: 0xFC09_0044 (PWMSDN)   Access: Read/Write

	7	6	5	4	3	2	1	0
R	IF	IE	0	LVL	0	PWM7IN	PWM7IL	SDNEN
W	w1c		RESTART					
Reset:	0	0	0	0	0	0	0	0
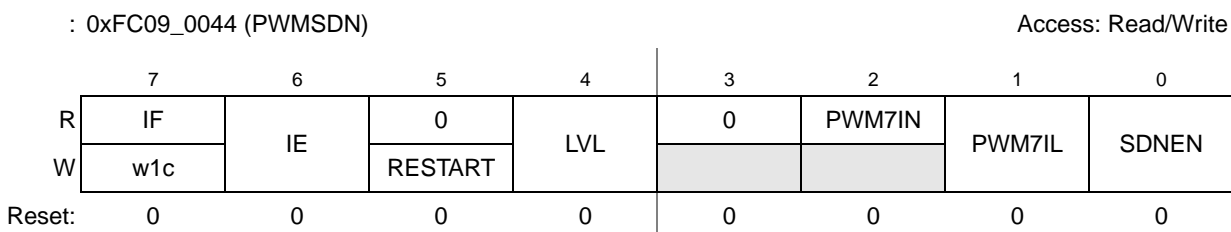
Figure 26-13. PWM Shutdown Register (PWMSDN)

Table 26-13. PWMSDN Field Descriptions

Field	Description
7 IF	PWM interrupt flag. Any change in state of PWM7IN is flagged by setting this bit. The flag is cleared by writing a 1 to it. Writing 0 has no effect. 0 No change in PWM7IN input 1 Change in PWM7IN input
6 IE	PWM interrupt enable. An interrupt is triggered to the device's interrupt controller when PWMSDN[IF] is set. 0 Interrupt is disabled 1 Interrupt is enabled
5 RESTART	PWM restart. After setting the RESTART bit, the PWM channels start running after the corresponding counter resets to zero. Also, if emergency shutdown is cleared (after being set), the PWM outputs restart after the corresponding counter resets to zero. This bit is self-clearing, so is always read as zero.
4 LVL	PWM shutdown output level. Describes the behavior of the PWM outputs when PWM7IN input is asserted and PWMSDN[SDNEN] is set. 0 PWM outputs are forced to logic 0 1 PWM outputs are forced to logic 1
3	Reserved, must be cleared.
2 PWM7IN	PWM channel 7 input status. Reflects the current status of the PWMOUT7 pin. Read only.
1 PWM7IL	PWM channel 7 input polarity. If PWMSDN[SDNEN] is set, this bit sets the active level of the PWM 7 channel 0 PWM 7 input is active low 1 PWN 7 input is active high
0 SDNEN	PWM emergency shutdown enable. If set, the pin associated with PWM channel 7 is forced to input and the emergency shutdown feature is enabled. 0 Emergency shutdown is disabled 1 Emergency shutdown is enabled

26.3   Functional Description

26.3.1   PWM Clock Select

There are four available clocks—clock A, B, SA (scaled A), and SB (scaled B)—all based on the internal bus clock.

Clock A and B can be programmed to run at 1, 1/2,..., 1/128 times the internal bus clock. Clock SA and SB use clock A and B respectively as an input and divide it further with a reloadable counter. The rates available for clock SA and SB are programmable to run at clock A and B divided by 2, 4,..., or 512. Each PWM channel has the capability of selecting one of two clocks, the prescaled clock (clock A or B) or the scaled clock (clock SA or SB). The block diagram in Figure 26-14 shows the four different clocks and how the scaled clocks are created.
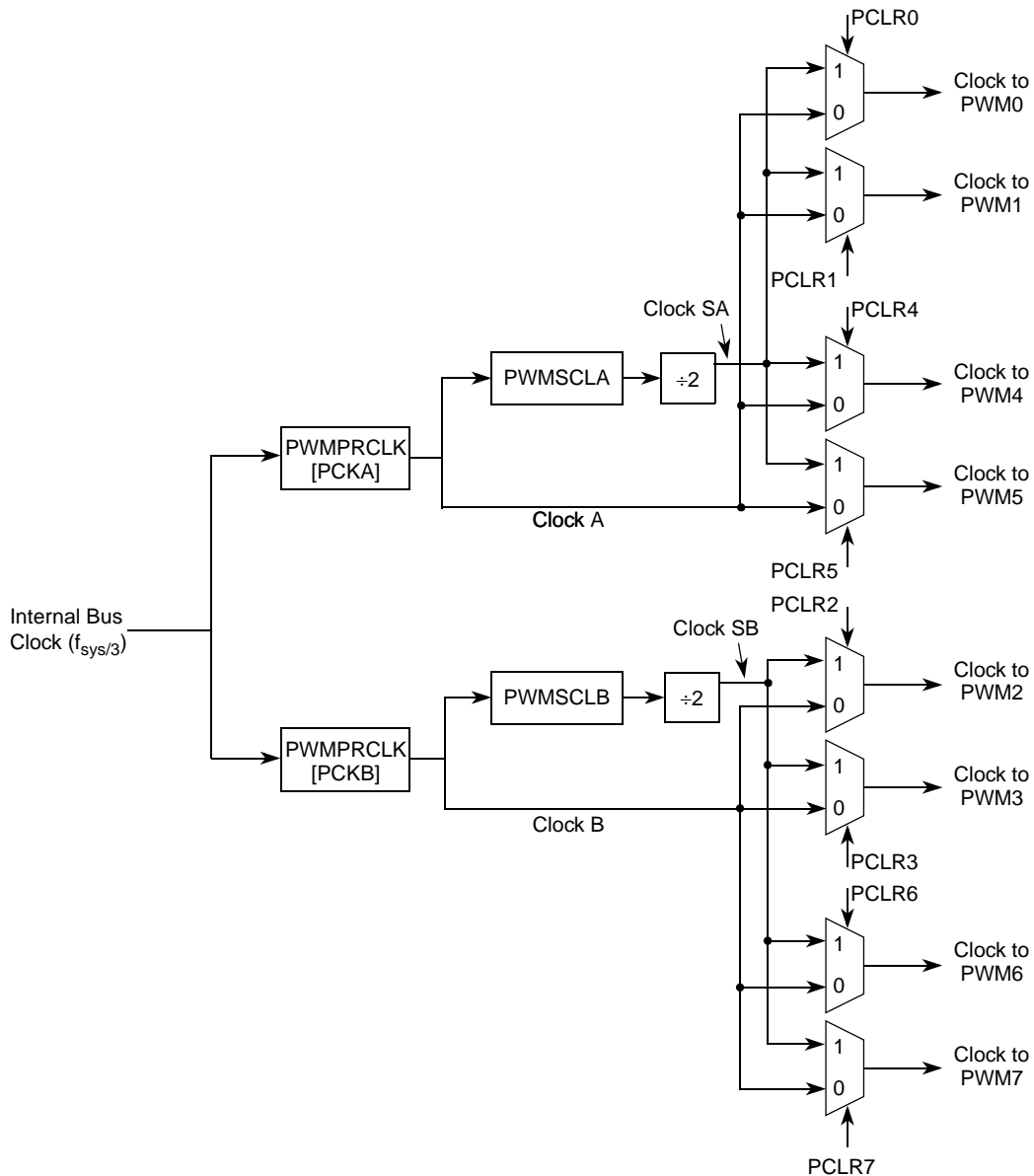
Figure 26-14. PWM Clock Select Block Diagram

26.3.1.1   Prescaled Clock (A or B)

The internal bus clock is the input clock to the PWM prescaler that can be disabled when the device is in debug mode by setting the PWMCTL[PFRZ] bit. This is useful for reducing power consumption and for emulation to freeze the PWM. The input clock is also disabled when all PWM channels are disabled (PWMEn=0).

Clock A and B are scaled values of the input clock. The value is software selectable for clock A and B and has options of 1, 1/2,..., or 1/128 times the internal bus clock. The value selected for clock A and B is determined by the PWMPRCLK[PCKAn] and PWMPRCLK[PCKBn] bits.

26.3.1.2   Scaled Clock (SA or SB)

The scaled A (SA) and scaled B (SB) clocks use clock A and B respectively as inputs, divide it further with a user programmable value, then divide this by 2. The rates available for clock SA are programmable to run at clock A divided by 2, 4,..., or 512. Similar rates are available for clock SB.

Clock SA equals clock A divided by two times the value in the PWMSCLA register:

$$\text{Clock SA} = \frac{\text{Clock A}}{2 \times \text{PWMSCLA}}$$

Eqn. 26-5

Similarly, clock SB is generated according to the following equation:

$$\text{Clock SB} = \frac{\text{Clock B}}{2 \times \text{PWMSCLB}}$$

Eqn. 26-6

As an example, consider the case in which the user writes 0xFF into the PWMSCLA register. Clock A for this case is selected to be internal bus clock divided by 4. A pulse occurs at a rate of once every 255×4 bus cycles. Passing this through the divide by two circuit produces a clock signal of the internal bus clock divided by 2040. Similarly, a value of 0x01 in the PWMSCLA register when clock A is internal bus clock divided by 4 produces an internal bus clock divided by 8 rate.

Writing to PWMSCLA or PWMSCLB causes the associated 8-bit down counter to be re-loaded. Otherwise, when changing rates, the counter would have to count down to 0x01 before counting at the proper rate. Forcing the associated counter to re-load the scale register value every time PWMSCLA or PWMSCLB is written prevents this.

Writing to the scale registers while channels are operating can cause irregularities in the PWM outputs.

26.3.1.3   Clock Select

Each PWM channel has the capability of selecting one of two clocks. For channels 0, 1, 4, and 5 the clock choices are clock A or SA. For channels 2, 3, 6 and 7, the choices are clock B or SB. The clock selection is done with the PWMCLK[PCLK*x*] control bits.

Changing clock control bits while channels are operating can cause irregularities in the PWM outputs.

26.3.2   PWM Channel Timers

The main part of the PWM module is the actual timers. Each of the timer channels has a counter, a period register, and a duty register (each are 8-bit). The waveform output period is controlled by a match between the period register and the value in the counter. The duty is controlled by a match between the duty register and the counter value and causes the state of the output to change during the period. The starting polarity of the output is also selectable on a per channel basis. Figure 26-15 shows a block diagram for a PWM timer.
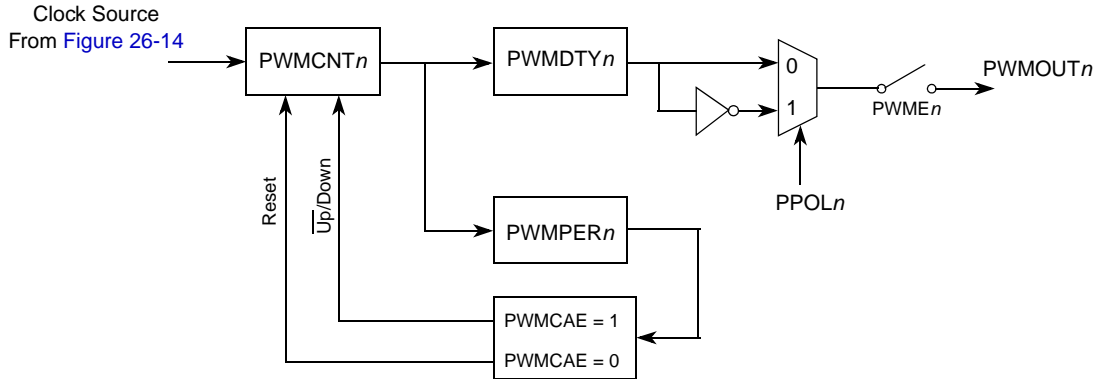
Figure 26-15. PWM Timer Channel Block Diagram

26.3.2.1   PWM Enable

Each PWM channel has an enable bit (PWME*n*) to start its waveform output. When any of the PWME*n* bits are set (PWME*n*=1), the associated PWM output signal is enabled immediately. However, the actual PWM waveform is not available on the associated PWM output until its clock source begins its next cycle; this is due to the synchronization of PWME*n* and the clock source. An exception is when channels are concatenated. Refer to Section 26.3.2.7, "PWM 16-Bit Functions" for more detail.

The first PWM cycle after enabling the channel can be irregular. When the channel is disabled (PWME*n*=0), the counter for the channel does not count.

26.3.2.2   PWM Polarity

Each channel has a polarity bit to allow starting a waveform cycle with a high or low signal. This is shown on the block diagram as a mux select. When one of the bits in the PWMPOL register is set, the associated PWM channel output is high at the beginning of the waveform, then goes low when the duty count is reached. Conversely, if the polarity bit is zero, the output starts low and then goes high when the duty count is reached.

26.3.2.3   PWM Period and Duty

Dedicated period and duty registers exist for each channel and are double buffered so that if they change while the channel is enabled, the change does not take effect until one of the following occurs:

- The effective period ends
- The PWMCNT*n* register is written (counter resets to 0x00)
- The channel is disabled, PWME*n* = 0

In this way, the output of the PWM is always the old waveform or the new waveform, not some variation in between. If the channel is not enabled, writes to the period and duty registers go directly to the latches as well as the buffer.

A change in duty or period can be forced into effect immediately by writing the new value to the duty and/or period registers and then writing to the counter. This forces the counter to reset and the new duty

and/or period values to be latched. In addition, because the counter is readable, it is possible to know where the count is with respect to the duty value, and software can be used to make adjustments. When forcing a new period or duty into effect immediately, an irregular PWM cycle can occur.

Depending on the polarity bit, the duty registers contain the count of the high time or the low time.

26.3.2.4 PWM Timer Counters

Each channel has a dedicated 8-bit up/down counter that runs at the rate of the selected clock source (see Figure 26-14 for the available clock sources and rates). The counter compares to two registers, a duty register and a period register, as shown in Figure 26-15. When the PWM counter matches the duty register, the output flip-flop changes state, causing the PWM waveform to also change state. A match between the PWM counter and the period register behaves differently depending on what output mode is selected as shown in Figure 26-15 and described in Section 26.3.2.5, "Left-Aligned Outputs" and Section 26.3.2.6, "Center-Aligned Outputs."

Each channel counter can be read at anytime without affecting the count or the operation of the PWM channel.

Any value written to the counter causes the counter to reset to 0x00, the counter direction to be set to up, the immediate load of duty and period registers with values from the buffers, and the output to change according to the polarity bit. When the channel is disabled (PWMEn = 0), the counter stops. When a channel becomes enabled (PWMEn = 1), the associated PWM counter continues from the count in the PWMCNTn register. This allows the waveform to continue where it left off when the channel is re-enabled. When the channel is disabled, writing 0 to the period register causes the counter to reset on the next selected clock.

NOTE

If the user wants to start a new clean PWM waveform without any history from the old waveform, the user must write to channel counter (PWMCNTn) prior to enabling the PWM channel (PWMEn = 1).

Generally, writes to the counter are done prior to enabling a channel to start from a known state. However, writing a counter can also be done while the PWM channel is enabled (counting). The effect is similar to writing the counter when the channel is disabled, except that the new period is started immediately with the output set according to the polarity bit. Writing to the counter while the channel is enabled can cause an irregular PWM cycle to occur.

The counter is cleared at the end of the effective period (see Section 26.3.2.5, "Left-Aligned Outputs" and Section 26.3.2.6, "Center-Aligned Outputs" for more details).

Table 26-14. PWM Timer Counter Conditions

Counter Clears (0x00)	Counter Counts	Counter Stops
When PWMCNTn register written to any value	When PWM channel is enabled (PWMEn = 1). Counts from last value in PWMCNTn.	When PWM channel is disabled (PWMEn = 0)
Effective period ends		

26.3.2.5 Left-Aligned Outputs

The PWM timer provides the choice of two types of outputs: left- or center-aligned. They are selected with the PWMCAE[CAEn] bits. If the CAEn bit is cleared, the corresponding PWM output is left-aligned.

In left-aligned output mode, the 8-bit counter is configured as an up counter only. It compares to two registers, a duty register and a period register, as shown in the block diagram in Figure 26-15. When the PWM counter matches the duty register, the output flip-flop changes state causing the PWM waveform to also change state. A match between the PWM counter and the period register resets the counter and the output flip-flop, as shown in Figure 26-16, as well as performing a load from the double buffer period and duty register to the associated registers, as described in Figure 26.3.2.3. The counter counts from 0 to the value in the period register minus 1.

NOTE

Changing the PWM output mode from left-aligned to center-aligned output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.
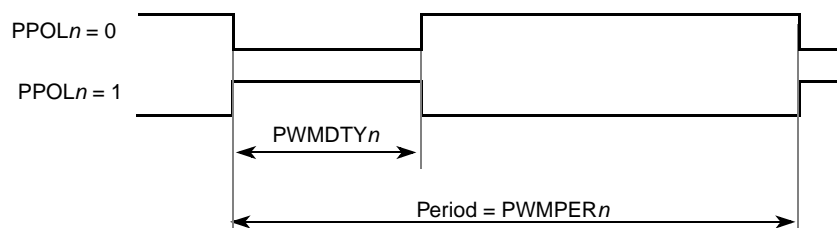
Figure 26-16. PWM Left-Aligned Output Waveform

To calculate the output frequency in left-aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by the value in the period register for that channel.

$$\text{PWM}n \text{ frequency} = \frac{\text{Clock (A, B, SA, or SB)}}{\text{PWMPER}n}$$

Eqn. 26-7

The PWMn duty cycle (high time as a percentage of period) is expressed as:

$$\text{Duty Cycle} = \left(1 - \text{PWMPOL[PPOL}n] - \frac{\text{PWMDTY}n}{\text{PWMPER}n}\right) \times 100\%$$

Eqn. 26-8

26.3.2.5.1 Left-Aligned Output Example

As an example of a left-aligned output, consider the following case:

> Clock source = internal bus clock, where internal bus clock = 80 MHz (12.5 ns period)
>
> PPOLn = 0, PWMPERn = 4, PWMDTYn = 1
>
> PWMn frequency = 80 MHz ÷ 4 = 20 MHz
>
> PWMn period = 50 ns
>
> PWMn Duty Cycle $= \left(1 - \frac{1}{4}\right) \times 100\% = 75\%$

The output waveform generated is below:
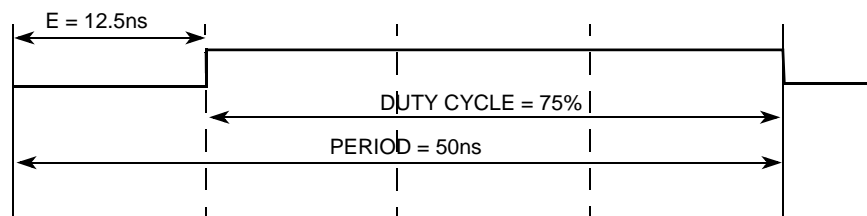
Figure 26-17. PWM Left-Aligned Output Example Waveform

26.3.2.6 Center-Aligned Outputs

For center-aligned output mode selection, set the PWMCAE[CAE*n*] bit and the corresponding PWM output is center-aligned.

The 8-bit counter operates as an up/down counter in this mode and is set to up when the counter is equal to 0x00. The counter compares to two registers, a duty register and a period register, as shown in the block diagram in Figure 26-15. When the PWM counter matches the duty register, the output flip-flop changes state, causing the PWM waveform to also change state. A match between the PWM counter and the period register changes the counter direction from an up-count to a down-count. When the PWM counter decrements and matches the duty register again, the output flip-flop changes state causing the PWM output to also change state. When the PWM counter decrements and reaches zero, the counter direction changes from a down-count back to an up-count, and a load from the double buffer period and duty registers to the associated registers is performed as described in Figure 26.3.2.3. The counter counts from 0 up to the value in the period register and then back down to 0. Thus the effective period is PWMPER*n* × 2.

Changing the PWM output mode from left-aligned output to center-aligned output (or vice versa) while channels are operating can cause irregularities in the PWM output. It is recommended to program the output mode before enabling the PWM channel.
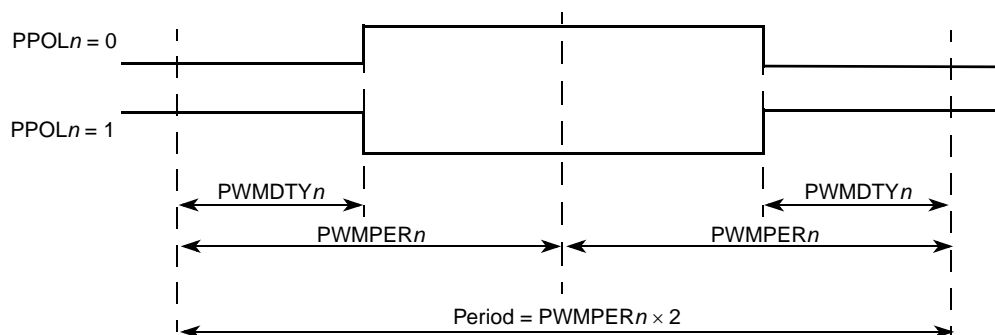
Figure 26-18. PWM Center-Aligned Output Waveform

To calculate the output frequency in center-aligned output mode for a particular channel, take the selected clock source frequency for the channel (A, B, SA, or SB) and divide it by twice the value in the period register for that channel.

$$\text{PWM}n \text{ frequency} = \frac{\text{Clock (A, B, SA, or SB)}}{2 \times \text{PWMPER}n} \qquad \textit{Eqn. 26-9}$$

The PWM*n* duty cycle (high time as a percentage of period) is expressed as:

$$\text{Duty Cycle} = \left(1 - \text{PWMPOL}[\text{PPOL}n] - \frac{\text{PWMDTY}n}{\text{PWMPER}n}\right) \times 100\%$$
Eqn. 26-10

26.3.2.6.1 Center-Aligned Output Example

As an example of a center-aligned output, consider the following case:

Clock source = internal bus clock, where internal bus clock = 80 MHz (12.5 ns period)

PPOLn = 0, PWMPERn = 4, PWMDTYn = 1

PWMn frequency = 80 MHz / (2×4) = 10 MHz

PWMn period = 100 ns

PWMn Duty Cycle $= \left(1 - \frac{1}{4}\right) \times 100\% = 75\%$

Shown below is the generated output waveform.
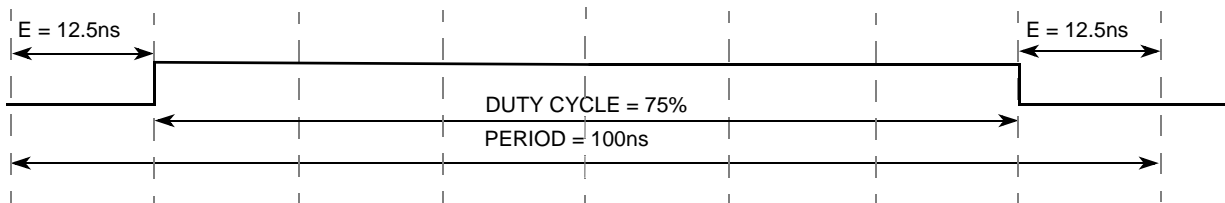
Figure 26-19. PWM Center-Aligned Output Example Waveform

26.3.2.7 PWM 16-Bit Functions

The PWM timer also has the option of generating 48-bit channels or four 16-bit channels for greater PWM resolution. This 16-bit channel option is achieved through the concatenation of two 8-bit channels.

The PWMCTL register contains four concatenation control bits, each used to concatenate a pair of PWM channels into one 16-bit channel. Channels 0 and 1 are concatenated with the CON01 bit, channels 2 and 3 are concatenated with the CON23 bit, and so on. Change these bits only when both corresponding channels are disabled.

As shown in Figure 26-20, when channels 2 and 3 are concatenated, channel 2 registers become the high order bytes of the double byte channel. When channels 0 and 1 are concatenated, channel 0 registers become the high order bytes of the double byte channel.

When using the 16-bit concatenated mode, the clock source is determined by the low order 8-bit channel clock select control bits (the odd numbered channel). The resulting PWM is output to the pins of the corresponding low order 8-bit channel, as shown in Figure 26-20. The polarity of the resulting PWM output is controlled by the PPOLn bit of the corresponding low order 8-bit channel as well.

After concatenated mode is enabled (PWMCTL[CONnn] bits set), enabling/disabling the corresponding 16-bit PWM channel is controlled by the low order PWMEn bit. In this case, the high order bytes' PWMEn bits have no effect, and their corresponding PWM output is disabled.

In concatenated mode, writes to the 16-bit counter by using a 16-bit access or writes to the low or high order byte of the counter resets the 16-bit counter. Reads of the 16-bit counter must be made by 16-bit access to maintain data coherency.
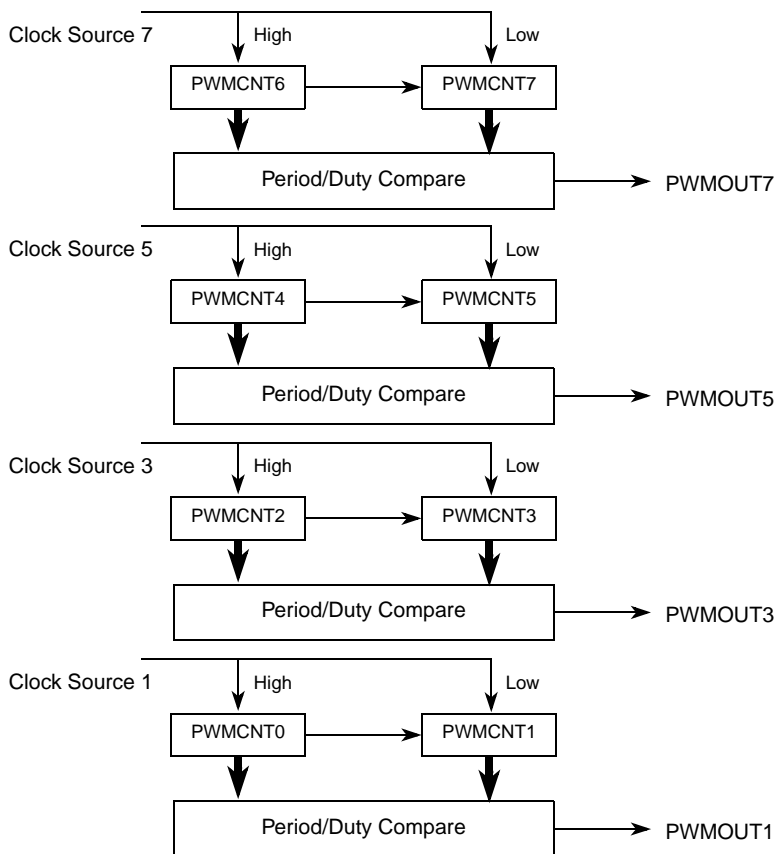
Figure 26-20. PWM 16-Bit Mode

Left- or center-aligned output mode can be used in concatenated mode and is controlled by the low order CAE*n* bit. The high order CAE*n* bit has no effect. The table shown below is used to summarize which channels are used to set the various control bits when in 16-bit mode.

Table 26-15. 16-bit Concatenation Mode Summary

CON*nn*	PWME*n*	PPOL*n*	PCLK*n*	CAE*n*	PWM*n* Output
CON67	PWM7	PPOL7	PCLK7	CAE7	PWMOUT7
CON45	PWM5	PPOL5	PCLK5	CAE5	PWMOUT5
CON23	PWME3	PPOL3	PCLK3	CAE3	PWMOUT3
CON01	PWME1	PPOL1	PCLK1	CAE1	PWMOUT1

26.3.2.8 PWM Boundary Cases

The following table summarizes the boundary conditions for the PWM regardless of the output mode (left- or center-aligned) and 8-bit (normal) or 16-bit (concatenation):

Table 26-16. PWM Boundary Cases

PWMDTY*n*	PWMPER*n*	PPOL*n*	PWM*n* Output
0x00 (indicates no duty)	>0x00	1	Always Low
0x00 (indicates no duty)	>0x00	0	Always High
XX	0x00[1] (indicates no period)	1	Always High
XX	0x00[1] (indicates no period)	0	Always Low
≥ PWMPER*n*	XX	1	Always High
≥ PWMPER*n*	XX	0	Always Low

[1] Counter = 0x00 and does not count.

Chapter 27
Watchdog Timer Module

27.1 Introduction

The watchdog timer (WDT) is a 16-bit timer used to help software recover from runaway code. The watchdog timer has a free-running down-counter (watchdog counter) that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown by servicing the watchdog.

27.1.1 Low-Power Mode Operation

This subsection describes the operation of the watchdog module in low-power modes and halted mode of operation (by issuing a HALT instruction). Low-power modes are described in Chapter 8, "Power Management." Table 27-1 shows the watchdog module operation in the low-power modes, and shows how this module may facilitate exit from each mode.

Table 27-1. Watchdog Module Operation in Low-power Modes

Low-power Mode	Watchdog Operation	Mode Exit
Wait	Normal if WCR[WAIT] cleared, stopped otherwise	Upon Watchdog reset
Doze	Normal if WCR[DOZE] cleared, stopped otherwise	Upon Watchdog reset
Stop	Stopped	No, only via external interrupts

In wait mode, with the watchdog control register's WAIT bit (WCR[WAIT]) set, watchdog timer operation stops. In wait mode with the WCR[WAIT] bit cleared, the watchdog timer continues to operate normally. In doze mode with the WCR[DOZE] bit set, the watchdog timer module operation stops. In doze mode with the WCR[DOZE] bit cleared, the watchdog timer continues to operate normally. Watchdog timer operation stops in stop mode. When stop mode is exited, the watchdog timer continues to operate in its pre-stop mode state.

In halted mode (entered by issuing a HALT instruction or asserting $\overline{\text{BKPT}}$) with the WCR[HALTED] bit set, watchdog timer module operation stops. When halted mode is exited, watchdog timer operation continues from the state it was in before entering halted mode, but any updates made in halted mode remain. If the WCR[HALTED] bit is cleared, the watchdog timer continues to operate normally after executing a HALT instruction. This is a debug feature available for the user

27.1.2   Block Diagram
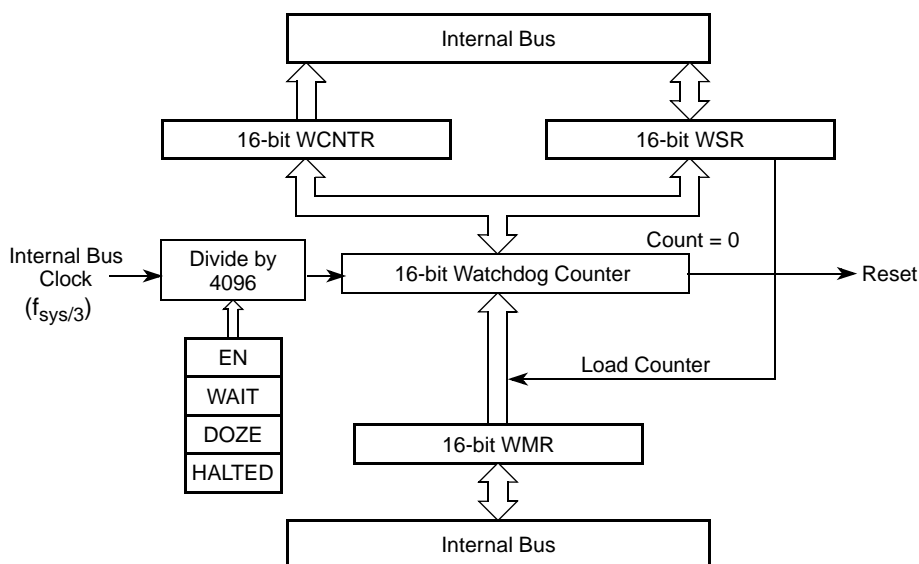
Figure 27-1. Watchdog Timer Block Diagram

27.2   Memory Map/Register Definition

This subsection describes the memory map and registers for the watchdog timer. Refer to Table 27-2 for an overview of the watchdog memory map.

NOTE

Longword accesses to any of the watchdog timer registers result in a bus error. Only byte and word accesses are allowed.

Table 27-2. Watchdog Timer Module Memory Map

Address[1]	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Only Access					
0xFC09_8000	Watchdog Control Register (WCR)	16	R/W	0x000F	27.2.1/27-3
0xFC09_8002	Watchdog Modulus Register (WMR)	16	R/W	0xFFFF	27.2.2/27-4
Supervisor/User Access					
0xFC09_8004	Watchdog Count Register (WCNTR)	16	R	0xFFFF	27.2.3/27-4
0xFC09_8006	Watchdog Service Register (WSR)	16	R/W	0x0000	27.2.4/27-5

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

27.2.1 Watchdog Control Register (WCR)

The 16-bit WCR configures watchdog timer operation.

Address: 0xFC09_8000   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	0	0	0	0	0	0	0	0	WAIT	DOZE	HALTED	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 27-2. Watchdog Control Register (WCR)

Table 27-3. WCR Field Descriptions

Field	Description
15–4	Reserved, should be cleared.
3 WAIT	Wait mode bit. Controls the function of the watchdog timer in wait mode. After written, the WAIT bit is not affected by further writes except in halted mode. Reset sets WAIT. 0 Watchdog timer not affected in wait mode 1 Watchdog timer stopped in wait mode
2 DOZE	Doze mode bit. Controls the function of the watchdog timer in doze mode. After written, the DOZE bit is not affected by further writes except in halted mode. Reset sets DOZE. 0 Watchdog timer not affected in doze mode 1 Watchdog timer stopped in doze mode
1 HALTED	Halted mode bit. Controls the function of the watchdog timer in halted/debug mode. After written, the HALTED bit is not affected by further writes except in halted mode. During halted mode, watchdog timer registers can be written and read normally. When halted mode is exited, timer operation continues from the state it was in before entering halted mode, but any updates made in halted mode remain. If a write-once register is written for the first time in halted mode, the register remains writable when halted mode is exited. 0 Watchdog timer not affected in halted mode 1 Watchdog timer stopped in halted mode **Note:** Changing the HALTED bit from 1 to 0 during halted mode starts the watchdog timer. Changing the HALTED bit from 0 to 1 during halted mode stops the watchdog timer.
0 EN	Watchdog enable bit. Enables the watchdog timer. After written, the EN bit is not affected by further writes except in halted mode. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. After disabled, the watchdog cannot be re-enabled. 0 Watchdog timer disabled 1 Watchdog timer enabled

27.2.2   Watchdog Modulus Register (WMR)

The WMR register determines the timer modulus reload value.

Address: 0xFC09_8002   Access: Supervisor
                                                                                                                    read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								WM								
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
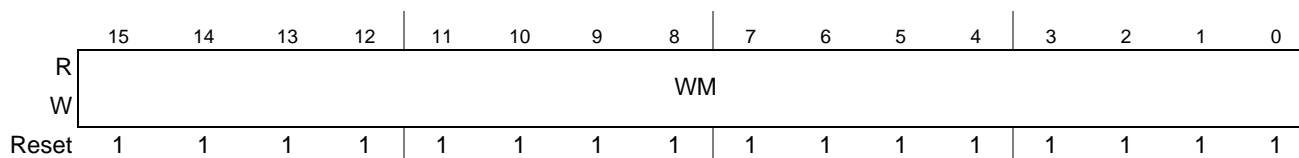
Figure 27-3. Watchdog Modulus Register (WMR)

Table 27-4. WMR Field Descriptions

Field	Description
15–0 WM	Watchdog modulus. Contains the modulus that is reloaded into the watchdog counter by a service sequence. After written, the WM[15:0] field is not affected by further writes except in halted mode. Writing to WMR immediately loads the new modulus value into the watchdog counter. The new value is also used at the next and all subsequent reloads. Reading WMR returns the value in the modulus register. Reset initializes the WM[15:0] field to 0xFFFF. **Note:** The prescaler counter is reset anytime a new value is loaded into the watchdog counter and also during reset.

27.2.3   Watchdog Count Register (WCNTR)

The WCNTR register provides visibility to the watchdog counter value. The timeout value for the watchdog timer is determined from the following equation:

$$\text{WDT timeout} = \frac{4096 \times \text{WCNTR}}{f_{sys/3}}$$ *Eqn. 27-1*

Thus, the maximum timeout is $4096 \times 2^{16} / f_{sys/3} = 3.36$ seconds at 80 MHz internal bus frequency.

Address: 0xFC09_8004   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								WC								
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
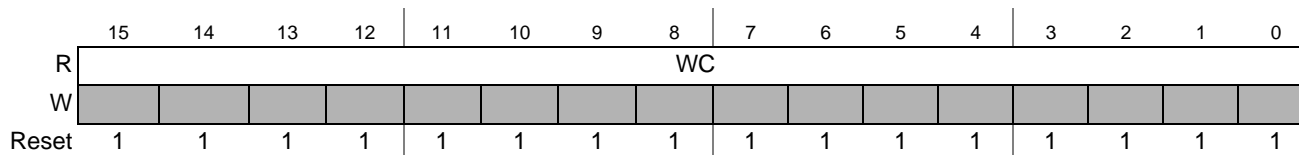
Figure 27-4. Watchdog Count Register (WCNTR)

Table 27-5. WCNTR Field Descriptions

Field	Description
15–0 WC	Watchdog count field. Reflects the current value in the watchdog counter. Reading the 16-bit WCNTR with two 8-bit reads is not guaranteed to return a coherent value. Writing to WCNTR has no effect, and write cycles are terminated normally.

27.2.4 Watchdog Service Register (WSR)

When the watchdog timer is enabled, writing 0x5555 and then 0xAAAA to WSR before the watchdog counter times out prevents a reset. If WSR is not serviced before the timeout, the watchdog timer sends a signal to the reset controller module that sets the RSR[WDR] bit and asserts a system reset.

Both writes must occur in the order listed before the timeout, but any number of instructions can be executed between the two writes. However, writing any value other than 0x5555 or 0xAAAA to WSR resets the servicing sequence, requiring both values to be written to keep the watchdog timer from causing a reset.

Address: 0xFC09_8006   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								WS								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
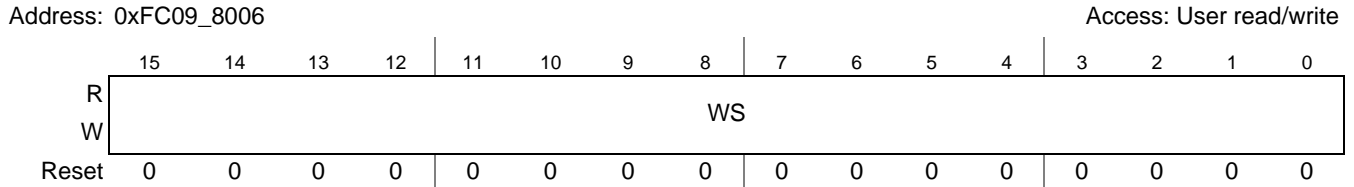
Figure 27-5. Watchdog Service Register (WSR)

Chapter 28
Programmable Interrupt Timers (PIT0–PIT3)

28.1 Introduction

This chapter describes the operation of the four programmable interrupt timer modules: PIT0–PIT3.

28.1.1 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.

28.1.2 Block Diagram
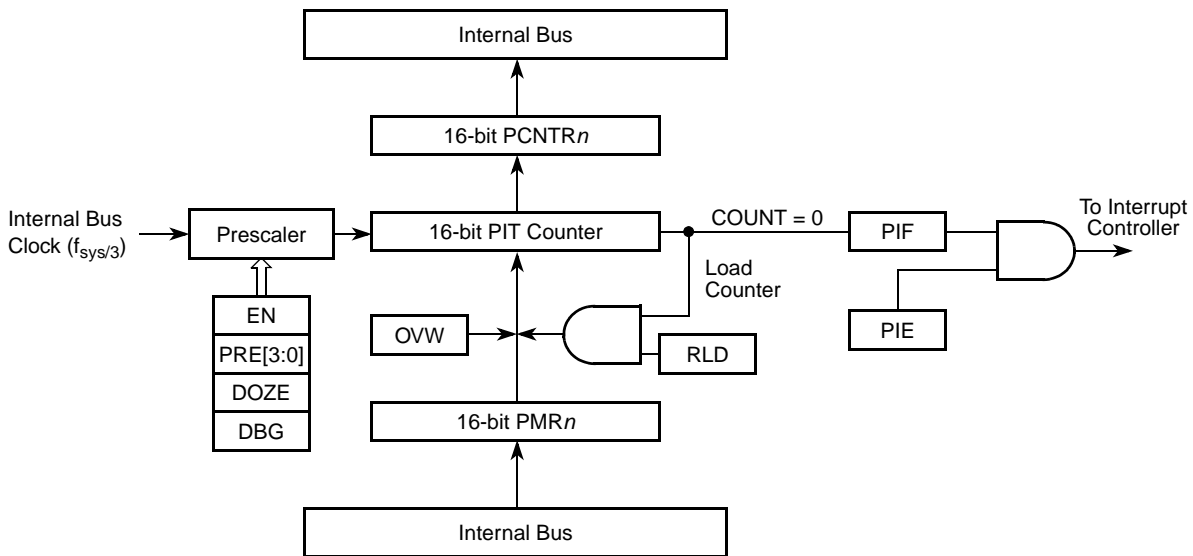
Figure 28-1. PIT Block Diagram

28.1.3 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 8, "Power Management." Table 28-1 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 28-1. PIT Module Operation in Low-power Modes

Low-power Mode	PIT Operation	Mode Exit
Wait	Normal	N/A
Doze	Normal if PCSRn[DOZE] cleared, stopped otherwise	Any interrupt at or above level in LPICR, exit doze mode if PCSRn[DOZE] is set. Otherwise interrupt assertion has no effect.
Stop	Stopped	No
Debug	Normal if PCSRn[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 PCSRn[DOZE] bit set, PIT module operation stops. In doze mode with the PCSRn[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 PCSRn[DBG] bit set, PIT module operation stops. In debug mode with the PCSRn[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.

28.2 Memory Map/Register Definition

This section contains a memory map (see Table 28-2) 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 28-2. Programmable Interrupt Timer Modules Memory Map

Address PIT 0 PIT 1 PIT 2 PIT 3	Register	Width (bits)	Access[1]	Reset Value	Section/Page
Supervisor Access Only Registers[2]					
0xFC08_0000 0xFC08_4000 0xFC08_8000 0xFC08_C000	PIT Control and Status Register (PCSRn)	16	R/W	0x0000	28.2.1/28-3

Table 28-2. Programmable Interrupt Timer Modules Memory Map (continued)

Address PIT 0 PIT 1 PIT 2 PIT 3	Register	Width (bits)	Access[1]	Reset Value	Section/Page
0xFC08_0002 0xFC08_4002 0xFC08_8002 0xFC08_C002	PIT Modulus Register (PMR*n*)	16	R/W	0xFFFF	28.2.2/28-5
User/Supervisor Access Registers					
0xFC08_0004 0xFC08_4004 0xFC08_8004 0xFC08_C004	PIT Count Register (PCNTR*n*)	16	R	0xFFFF	28.2.3/28-5

[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.

28.2.1 PIT Control and Status Register (PCSR*n*)

The PCSR*n* registers configure the corresponding timer's operation.

Address: 0xFC08_0000 (PCSR0)   Access: Supervisor
     0xFC08_4000 (PCSR1)   read/write
     0xFC08_8000 (PCSR2)
     0xFC08_C000 (PCSR3)

	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
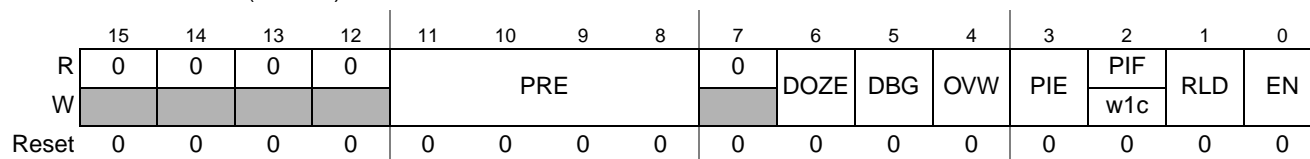
Figure 28-2. PCSR*n* Register

MCF5329 Reference Manual, Rev 3

Freescale Semiconductor   28-3

Table 28-3. PCSR*n* Field Descriptions

Field	Description
15–12	Reserved, must be cleared.
11–8 PRE	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. | | PRE	Internal Bus Clock Divisor	Decimal Equivalent
---	---	---
0000	2^0	1
0001	2^1	2
0010	2^2	4
...	...	...
1101	2^{13}	8192
1110	2^{14}	16384
1111	2^{15}	32768	
7	Reserved, must be cleared.
6 DOZE	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.
5 DBG	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 **Note:** 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.
4 OVW	Overwrite. Enables writing to PMR*n* to immediately overwrite the value in the PIT counter. 0 Value in PMR*n* replaces value in PIT counter when count reaches 0x0000. 1 Writing PMR*n* immediately replaces value in PIT counter.
3 PIE	PIT interrupt enable. This read/write bit enables PIF flag to generate interrupt requests. 0 PIF interrupt requests disabled 1 PIF interrupt requests enabled
2 PIF	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.

Table 28-3. 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

28.2.2 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								PM								
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
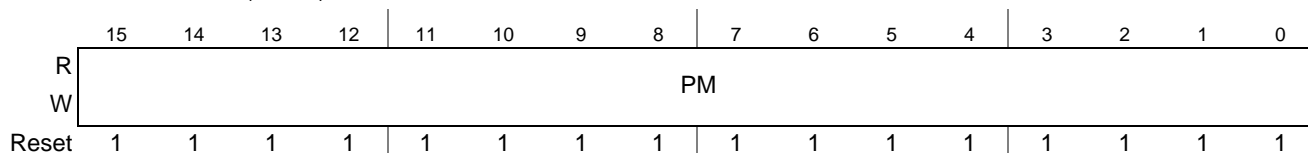
Figure 28-3. PIT Modulus Register (PMR*n*)

Table 28-4. 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.

28.2.3 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)   Access: User read only
        0xFC08_4004 (PCNTR1)
        0xFC08_8004 (PCNTR2)
        0xFC08_C004 (PCNTR3)

	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
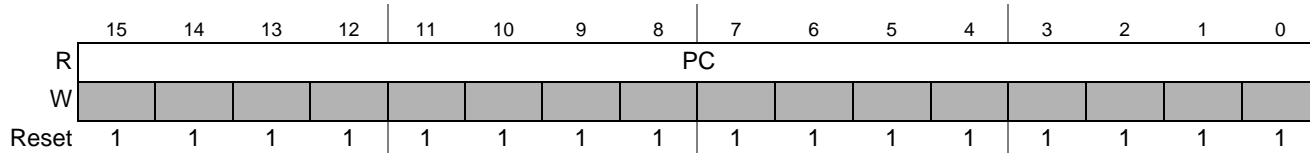
Figure 28-4. PIT Count Register (PCNTR*n*)

Table 28-5. 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.

28.3 Functional Description

This section describes the PIT functional operation.

28.3.1 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.
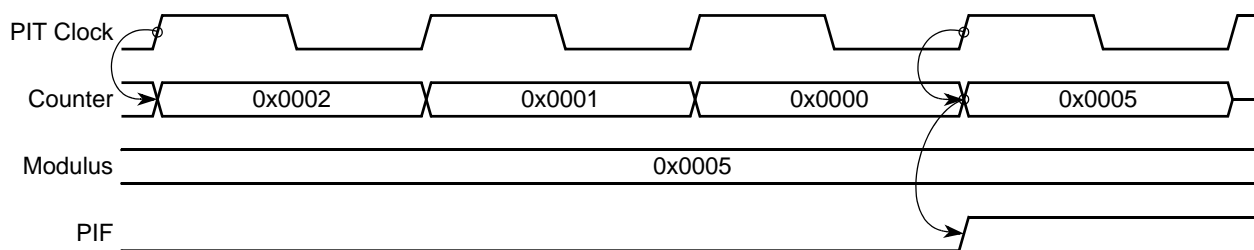
Figure 28-5. Counter Reloading from the Modulus Latch

28.3.2 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.
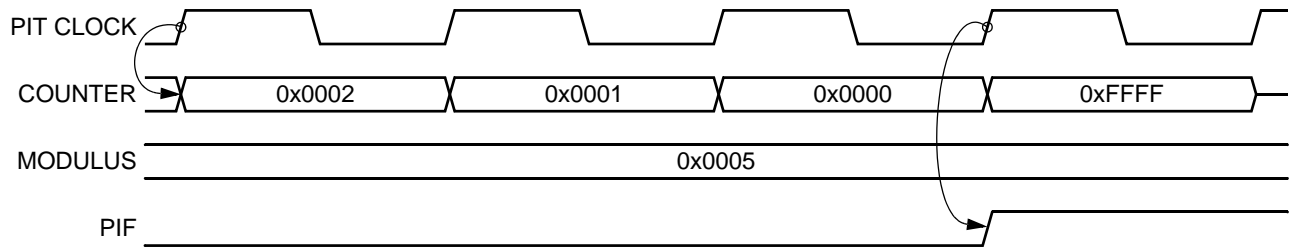
Figure 28-6. Counter in Free-Running Mode

28.3.3   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.

$$\text{Timeout period} = \frac{2^{\text{PCSRn[PRE]}} \times (\text{PMRn[PM]} + 1)}{f_{\text{sys/3}}}$$ *Eqn. 28-1*

28.3.4   Interrupt Operation

Table 28-6 shows the interrupt request generated by the PIT.

Table 28-6. 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 29
DMA Timers (DTIM0–DTIM3)

29.1 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.

29.1.1 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 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 13, "General Purpose I/O Module") prior to configuring the DMA Timers.

Figure 29-1 is a block diagram of one of the four identical timer modules.
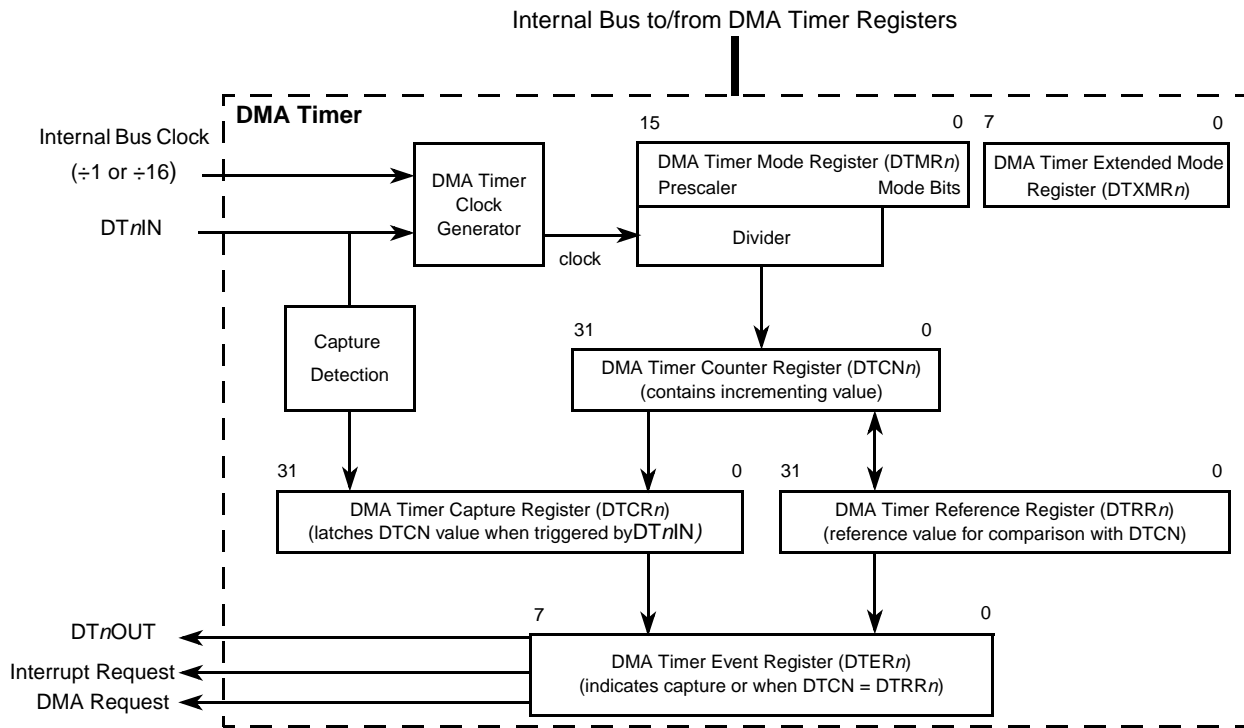
Figure 29-1. DMA Timer Block Diagram

29.1.2   Features

Each DMA timer module has:

- Maximum timeout period of 219,902 seconds at 80 MHz (~61 hours)
- 12.5-ns resolution at 80 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

29.2   Memory Map/Register Definition

The timer module registers, shown in Table 29-1, can be modified at any time.

Table 29-1. DMA Timer Module Memory Map

Address DMA Timer 0 DMA Timer 1 DMA Timer 2 DMA Timer 3	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC07_0000 0xFC07_4000 0xFC07_8000 0xFC07_C000	DMA Timer *n* Mode Register (DTMR*n*)	16	R/W	0x0000	29.2.1/29-3
0xFC07_0002 0xFC07_4002 0xFC07_8002 0xFC07_C002	DMA Timer *n* Extended Mode Register (DTXMR*n*)	8	R/W	0x00	29.2.2/29-4
0xFC07_0003 0xFC07_4003 0xFC07_8003 0xFC07_C003	DMA Timer *n* Event Register (DTER*n*)	8	R/W	0x00	29.2.3/29-5
0xFC07_0004 0xFC07_4004 0xFC07_8004 0xFC07_C004	DMA Timer *n* Reference Register (DTRR*n*)	32	R/W	0xFFFF_FFFF	29.2.4/29-6
0xFC07_0008 0xFC07_4008 0xFC07_8008 0xFC07_C008	DMA Timer *n* Capture Register (DTCR*n*)	32	R/W	0x0000_0000	29.2.5/29-7
0xFC07_000C 0xFC07_400C 0xFC07_800C 0xFC07_C00C	DMA Timer *n* Counter Register (DTCN*n*)	32	R	0x0000_0000	29.2.6/29-8

29.2.1   DMA Timer Mode Registers (DTMR*n*)

DTMRs, shown in Figure 29-2, program the prescaler and various timer modes.

Address: 0xFC07_0000 (DTMR0)   Access: User read/write
         0xFC07_4000 (DTMR1)
         0xFC07_8000 (DTMR2)
         0xFC07_C000 (DTMR3)

	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
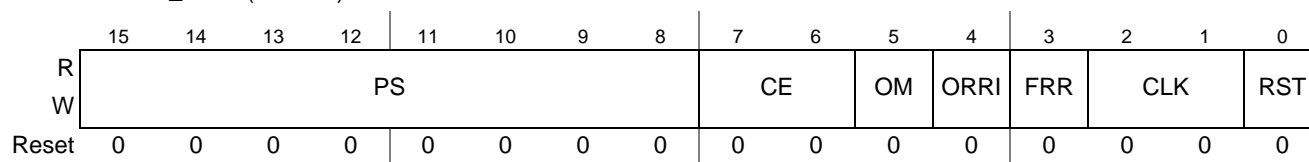
Figure 29-2. DTMR*n* Registers

Table 29-2. 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 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 (12.5-ns resolution at 80 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 slightly. 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

29.2.2 DMA Timer Extended Mode Registers (DTXMR*n*)

The DTXMR*n* register programs DMA request and increment modes for the timers.

Address: 0xFC07_0002 (DTXMR0)   Access: User read/write
          0xFC07_4002 (DTXMR1)
          0xFC07_8002 (DTXMR2)
          0xFC07_C002 (DTXMR3)

	7	6	5	4	3	2	1	0
R	DMAEN	0	0	0	0	0	0	MODE16
W								
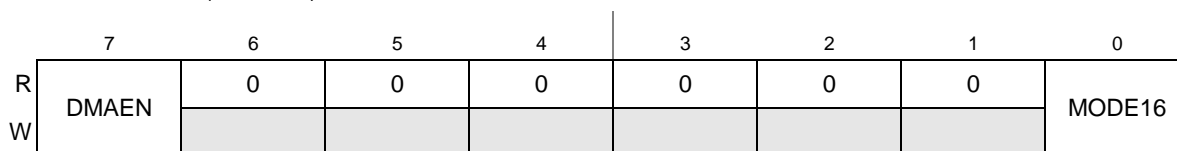
Figure 29-3. DTXMR*n* Registers

Table 29-3. 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
6–1	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

29.2.3   DMA Timer Event Registers (DTER*n*)

DTER*n*, shown in Figure 29-4, 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)   Access: User read/write
         0xFC07_4003 (DTER1)
         0xFC07_8003 (DTER2)
         0xFC07_C003 (DTER3)

	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 29-4. DTER*n* Registers

Table 29-4. DTER*n* Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1 REF	Output reference event. The counter value (DTCN*n*) equals DTRR*n*. Writing a 1 to REF clears the event condition. Writing a 0 has no effect.

REF	DTMR*n*[ORRI]	DTXMR*n*[DMAEN]	
0	X	X	No event
1	0	0	No request asserted
1	0	1	No request asserted
1	1	0	Interrupt request asserted
1	1	1	DMA request asserted

Field	Description
0 CAP	Capture event. The counter value has been latched into DTCR*n*. Writing a 1 to CAP clears the event condition. Writing a 0 has no effect.

CAP	DTMR*n*[CE]	DTXMR*n* [DMAEN]	
0	XX	X	No event
1	00	0	Disable capture event output
1	00	1	Disable capture event output
1	01	0	Capture on rising edge and trigger interrupt
1	01	1	Capture on rising edge and trigger DMA
1	10	0	Capture on falling edge and trigger interrupt
1	10	1	Capture on falling edge and trigger DMA
1	11	0	Capture on any edge and trigger interrupt
1	11	1	Capture on any edge and trigger DMA

29.2.4   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)   Access: User read/write
       0xFC07_4004 (DTRR1)
       0xFC07_8004 (DTRR2)
       0xFC07_C004 (DTRR3)

	31 30 29 28	27 26 25 24	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
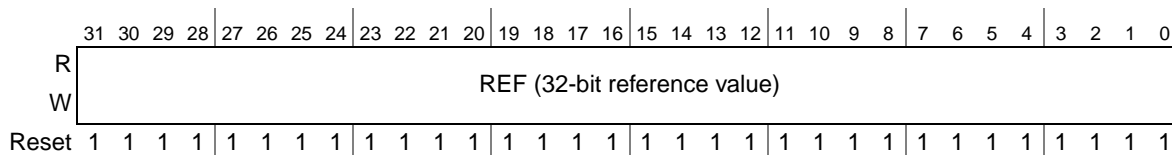
Figure 29-5. DTRR*n* Registers

Table 29-5. 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.

29.2.5 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)   Access: User read-only
       0xFC07_4008 (DTCR1)
       0xFC07_8008 (DTCR2)
       0xFC07_C008 (DTCR3)

	31 30 29 28	27 26 25 24	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
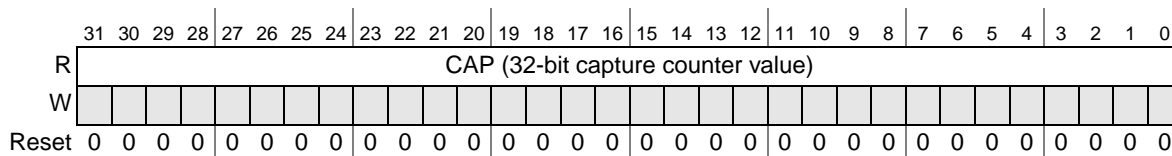
Figure 29-6. DTCR*n* Registers

Table 29-6. 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*.

29.2.6 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)   Access: User read/write
           0xFC07_400C (DTCN1)
           0xFC07_800C (DTCN2)
           0xFC07_C00C (DTCN3)

	31 30 29 28	27 26 25 24	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
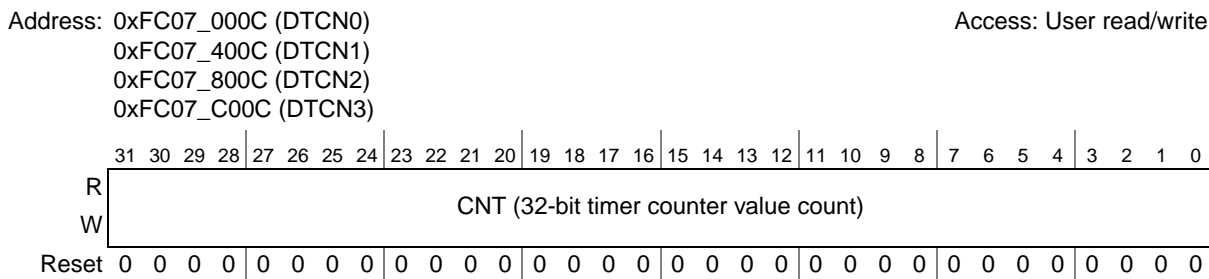
Figure 29-7. DMA Timer Counters (DTCN*n*)

Table 29-7. 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.

29.3 Functional Description

29.3.1 Prescaler

The prescaler clock input is selected from the internal bus clock ($f_{sys/3}$ 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*.

29.3.2 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.

29.3.3 Reference Compare

Each DMA timer can be configured to count up to a reference value, at which point 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.

29.3.4 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.

29.3.5 IEEE 1588 Support

The DMA timers on this device can use the Ethernet assembly's IEEE-1588 timebase count value as its clock source. This feature supports triggering events via processor interrupts or DMA requests based on network time values.

29.4 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.

29.4.1 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
```

29.4.2   Calculating Time-Out Values

Equation 29-1 determines time-out periods for various reference values:

$$\text{Timeout period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}n[\text{PS}] + 1) \times (\text{DTRR}n[\text{REF}] + 1) \qquad \textbf{\textit{Eqn. 29-1}}$$

When calculating time-out periods, add 1 to the prescaler to simplify calculating, because DTMRn[PS] equals 0x00 yields a prescaler of 1, and DTMRn[PS] equals 0xFF yields a prescaler of 256.

For example, if a 80-MHz timer clock is divided by 16, DTMRn[PS] equals 0x7F, and the timer is referenced at 0x1312C (78,124 decimal), the time-out period is:

$$\text{Timeout period} = \frac{1}{80 \times 10^6} \times 16 \times (127 + 1) \times (78124 + 1) = 2.00 \text{ s} \qquad \textit{Eqn. 29-2}$$

Chapter 30
Queued Serial Peripheral Interface (QSPI)

30.1 Introduction

This chapter describes the queued serial peripheral interface (QSPI) module.

30.1.1 Block Diagram

Figure 30-1 illustrates the QSPI module.
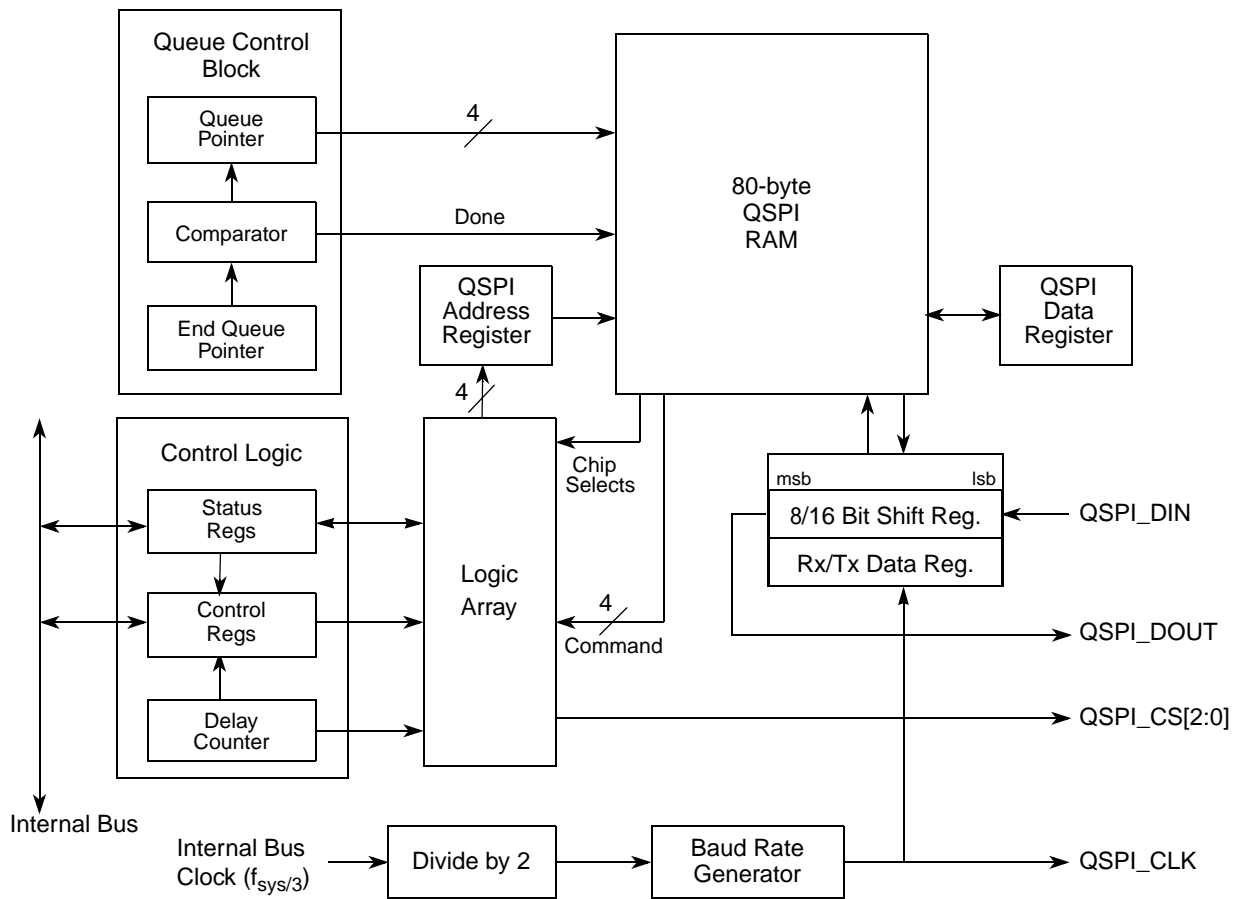
Figure 30-1. QSPI Block Diagram

30.1.2 Overview

The queued serial peripheral interface module provides a serial peripheral interface with queued transfer capability. It allows users to queue up to 16 transfers at once, eliminating CPU intervention between transfers. Transfer RAM in the QSPI is indirectly accessible using address and data registers.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 13, "General Purpose I/O Module") prior to configuring the QSPI module.

30.1.3 Features

Features include:

- Programmable queue to support up to 16 transfers without user intervention
  — 80 bytes of data storage provided
- Supports transfer sizes of 8 to 16 bits in 1-bit increments
- Three peripheral chip-select lines for control of up to 7 devices (All chip selects may not be available on all devices. See Chapter 2, "Signal Descriptions," for details on which chip-selects are pinned-out.)
- Baud rates from 156.9 Kbps to 20 Mbps at 80 MHz internal bus frequency
- Programmable delays before and after transfers
- Programmable QSPI clock phase and polarity
- Supports wraparound mode for continuous transfers

30.1.4 Modes of Operation

Because the QSPI module only operates in master mode, the master bit in the QSPI mode register (QMR[MSTR]) must be set for the QSPI to function properly. If the master bit is not set, QSPI activity is indeterminate. The QSPI can initiate serial transfers but cannot respond to transfers initiated by other QSPI masters.

30.2 External Signal Description

The module provides access to as many as 7 devices with a total of six signals: QSPI_DOUT, QSPI_DIN, QSPI_CLK, QSPI_CS[2:0].

Peripheral chip-select signals, QSPI_CS*n*, are used to select an external device as the source or destination for serial data transfer. Signals are asserted when a command in the queue is executed. More than one chip-select signal can be asserted simultaneously.

Although QSPI_CS*n* signals function as simple chip selects in most applications, up to 7 devices can be selected by decoding them with an external 3-to-8 decoder.

Table 30-1. QSPI Input and Output Signals and Functions

Signal Name	Hi-Z or Actively Driven	Function
Data output (QSPI_DOUT)	Configurable	Serial data output from QSPI
Data input (QSPI_DIN)	N/A	Serial data input to QSPI
Serial clock (QSPI_CLK)	Actively driven	Clock output from QSPI
Peripheral chip selects (QSPI_CS*n*)	Actively driven	Peripheral selects from QSPI

30.3 Memory Map/Register Definition

Table 30-2 is the QSPI register memory map. Reading reserved locations returns zeros.

Table 30-2. QSPI Memory Map

Address[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC05_C000	QSPI Mode Register (QMR)	16	R/W	0x0104	30.3.1/30-3
0xFC05_C004	QSPI Delay Register (QDLYR)	16	R/W	0x0404	30.3.2/30-5
0xFC05_C008	QSPI Wrap Register (QWR)	16	R/W[2]	0x0000	30.3.3/30-6
0xFC05_C00C	QSPI Interrupt Register (QIR)	16	R/W[2]	0x0000	30.3.4/30-6
0xFC05_C010	QSPI Address Register (QAR)	16	R/W[2]	0x0000	30.3.5/30-7
0xFC05_C014	QSPI Data Register (QDR)	16	R/W	0x0000	30.3.6/30-8

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion.

[2] See the register description for special cases. Some bits may be read- or write-only.

30.3.1 QSPI Mode Register (QMR)

The QMR, shown in Figure 30-2, determines the basic operating modes of the QSPI module. Parameters such as QSPI_CLK polarity and phase, baud rate, master mode operation, and transfer size are determined by this register.

NOTE

Because the QSPI does not operate in slave mode, the master mode enable bit (QMR[MSTR]) must be set for the QSPI module to operate correctly.

Address: 0xFC05_C000 (QMR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MSTR	0		BITS			CPOL	CPHA				BAUD				
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
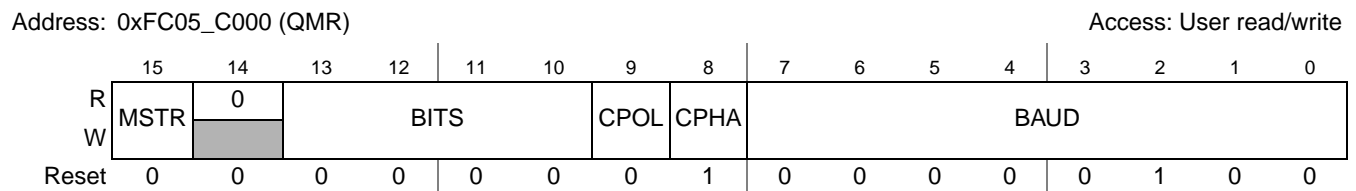
Figure 30-2. QSPI Mode Register (QMR)

Table 30-3. QMR Field Descriptions

Field	Description
15 MSTR	Master mode enable. 0 Reserved, do not use. 1 The QSPI is in master mode. Must be set for the QSPI module to operate correctly.
14	Reserved, must be cleared.
13–10 BITS	Transfer size. Determines the number of bits to be transferred for each entry in the queue. | BITS	Bits per Transfer | |---|---| | 0000 | 16 | | 0001–0111 | Reserved | | 1000 | 8 | | 1001 | 9 | | 1010 | 10 | | 1011 | 11 | | 1100 | 12 | | 1101 | 13 | | 1110 | 14 | | 1111 | 15 |
9 CPOL	Clock polarity. Defines the clock polarity of QSPI_CLK. 0 The inactive state value of QSPI_CLK is logic level 0. 1 The inactive state value of QSPI_CLK is logic level 1.
8 CPHA	Clock phase. Defines the QSPI_CLK clock-phase. 0 Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. 1 Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.
7–0 BAUD	Baud rate divider. The baud rate is selected by writing a value in the range 2–255. A value of zero disables the QSPI. A value of 1 is an invalid setting. The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression: $\quad$ QMR[BAUD] = $f_{sys/3}$ / (2 × [desired QSPI_CLK baud rate])

Figure 30-3 shows an example of a QSPI clocking and data transfer.
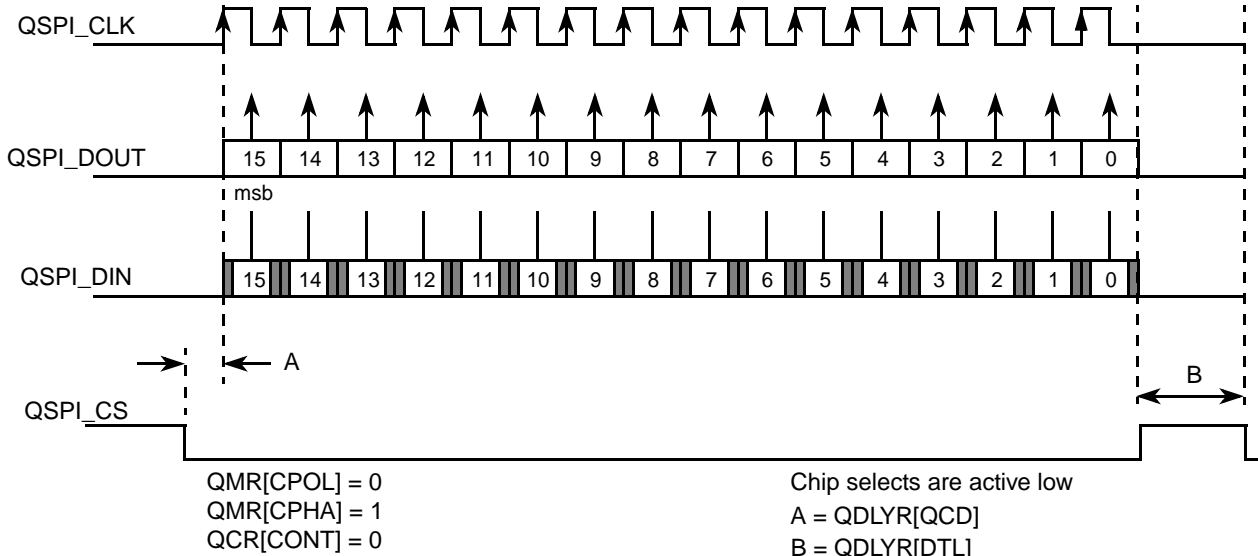
Figure 30-3. QSPI Clocking and Data Transfer Example

30.3.2 QSPI Delay Register (QDLYR)

The QDLYR is used to initiate master mode transfers and to set various delay parameters.

Address: 0xFC05_C004 (QDLYR)   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SPE			QCD								DTL				
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0

Figure 30-4. QSPI Delay Register (QDLYR)

Table 30-4. QDLYR Field Descriptions

Field	Description
15 SPE	QSPI enable. When set, the QSPI initiates transfers in master mode by executing commands in the command RAM. The QSPI clears this bit automatically when a transfer completes. The user can also clear this bit to abort transfer unless QIR[ABRTL] is set. The recommended method for aborting transfers is to set QWR[HALT].
14–8 QCD	QSPI_CLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition. See Section 30.4.3, "Transfer Delays" for information on setting this bit field.
7–0 DTL	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

30.3.3 QSPI Wrap Register (QWR)

The QSPI wrap register provides halt transfer control, wraparound settings, and queue pointer locations.

Address: 0xFC05_C008 (QWR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HALT	WREN	WRTO	CSIV	ENDQP				CPTQP				NEWQP			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
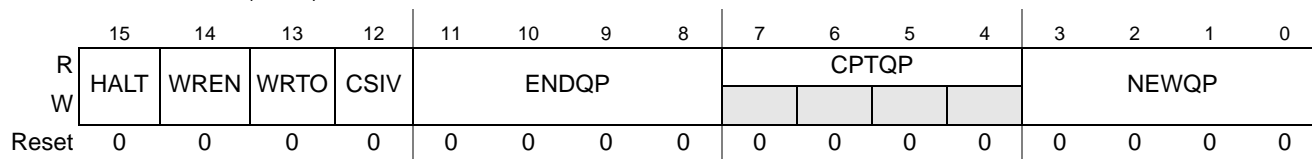
Figure 30-5. QSPI Wrap Register (QWR)

Table 30-5. QWR Field Descriptions

Field	Description
15 HALT	Halt transfers. Assertion of this bit causes the QSPI to stop execution of commands after it has completed execution of the current command.
14 WREN	Wraparound enable. Enables wraparound mode. 0 Execution stops after executing the command pointed to by QWR[ENDQP]. 1 After executing command pointed to by QWR[ENDQP], wrap back to entry zero, or the entry pointed to by QWR[NEWQP] and continue execution.
13 WRTO	Wraparound location. Determines where the QSPI wraps to in wraparound mode. 0 Wrap to RAM entry zero. 1 Wrap to RAM entry pointed to by QWR[NEWQP].
12 CSIV	QSPI_CS inactive level. 0 QSPI chip select outputs return to zero when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 0, chip selects are active high). 1 QSPI chip select outputs return to one when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 1, chip selects are active low).
11–8 ENDQP	End of queue pointer. Points to the RAM entry that contains the last transfer description in the queue.
7–4 CPTQP	Completed queue entry pointer. Points to the RAM entry that contains the last command to have been completed. This field is read only.
3–0 NEWQP	Start of queue pointer. This 4-bit field points to the first entry in the RAM to be executed on initiating a transfer.

30.3.4 QSPI Interrupt Register (QIR)

The QIR contains QSPI interrupt enables and status flags.

Address: 0xFC05_C00C (QIR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WCEFB	ABRTB	0	ABRTL	WCEFE	ABRTE	0	SPIFE	0	0	0	0	WCEF	ABRT	0	SPIF
W													w1c	w1c		w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
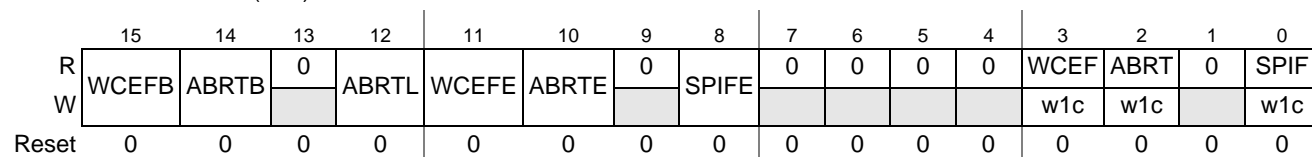
Figure 30-6. QSPI Interrupt Register (QIR)

Table 30-6. QIR Field Descriptions

Field	Description
15 WCEFB	Write collision access error enable. A write collision occurs during a data transfer when the RAM entry containing the current command is written to by the CPU with the QDR. When this bit is asserted, the write access to QDR results in an access error.
14 ABRTB	Abort access error enable. An abort occurs when QDLYR[SPE] is cleared during a transfer. When set, an attempt to clear QDLYR[SPE] during a transfer results in an access error.
13	Reserved, must be cleared.
12 ABRTL	Abort lock-out. When set, QDLYR[SPE] cannot be cleared by writing to the QDLYR. QDLYR[SPE] is only cleared by the QSPI when a transfer completes.
11 WCEFE	Write collision (WCEF) interrupt enable. 0 Write collision interrupt disabled 1 Write collision interrupt enabled
10 ABRTE	Abort (ABRT) interrupt enable. 0 Abort interrupt disabled 1 Abort interrupt enabled
9	Reserved, must be cleared.
8 SPIFE	QSPI finished (SPIF) interrupt enable. 0 SPIF interrupt disabled 1 SPIF interrupt enabled
7–4	Reserved, must be cleared.
3 WCEF	Write collision error flag. Indicates that an attempt has been made to write to the RAM entry that is currently being executed. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.
2 ABRT	Abort flag. Indicates that QDLYR[SPE] has been cleared by the user writing to the QDLYR rather than by completion of the command queue by the QSPI. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.
1	Reserved, must be cleared.
0 SPIF	QSPI finished flag. Asserted when the QSPI has completed all the commands in the queue. Set on completion of the command pointed to by QWR[ENDQP], and on completion of the current command after assertion of QWR[HALT]. In wraparound mode, this bit is set every time the command pointed to by QWR[ENDQP] is completed. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.

30.3.5   QSPI Address Register (QAR)

The QAR is used to specify the location in the QSPI RAM that read and write operations affect. As shown in Section 30.4.1, "QSPI RAM", the transmit RAM is located at addresses 0x0 to 0xF, the receive RAM is located at 0x10 to 0x1F, and the command RAM is located at 0x20 to 0x2F. (These addresses refer to the QSPI RAM space, not the device memory map.)

NOTE

A read or write to the QSPI RAM causes QAR to increment. However, the QAR does not wrap after the last queue entry within each section of the RAM. The application software must manage address range errors.

Address: 0xFC05_C010 (QAR)   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	0	0	0	0			ADDR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
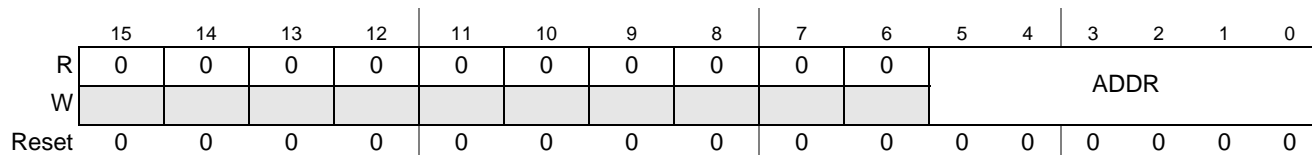
Figure 30-7. QSPI Address Register (QAR)

Table 30-7. QAR Field Descriptions

Field	Description
15–6	Reserved, must be cleared.
5–0 ADDR	Address used to read/write the QSPI RAM. Ranges are as follows: 0x00–0x0F Transmit RAM 0x10–0x1F Receive RAM 0x20–0x2F Command RAM 0x30–0x3F Reserved

30.3.6 QSPI Data Register (QDR)

The QDR is used to access QSPI RAM indirectly. The CPU reads and writes all data from and to the QSPI RAM through this register.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR]. This also causes the value in QAR to increment. Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

Address: 0xFC05_C014 (QDR)   Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R								DATA								
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
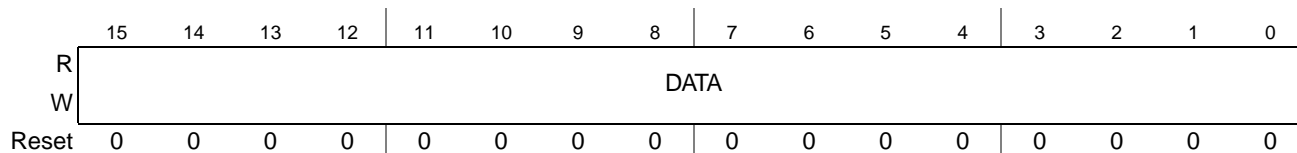
Figure 30-8. QSPI Data Register (QDR)

Table 30-8. QDR Field Descriptions

Field	Description
15–0 DATA	A write to this field causes data to be written to the QSPI RAM entry specified by QAR[ADDR]. Similarly, a read of this field returns the data in the QSPI RAM at the address specified by QAR[ADDR]. During command RAM accesses (QAR[ADDR] = 0x20–0x2F), only the most significant byte of this field is used.

30.3.7 Command RAM Registers (QCR0–QCR15)

The command RAM is accessed using the upper byte of the QDR; the QSPI cannot modify information in command RAM. There are 16 bytes in the command RAM. Each byte is divided into two fields. The chip select field enables external peripherals for transfer. The command field provides transfer operations.

NOTE

The command RAM is accessed only using the most significant byte of
QDR and indirect addressing based on QAR[ADDR].

Address: QAR[ADDR]   Access: CPU write-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																
W	CONT	BITSE	DT	DSCK	0		QSPI_CS		0	0	0	0	0	0	0	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
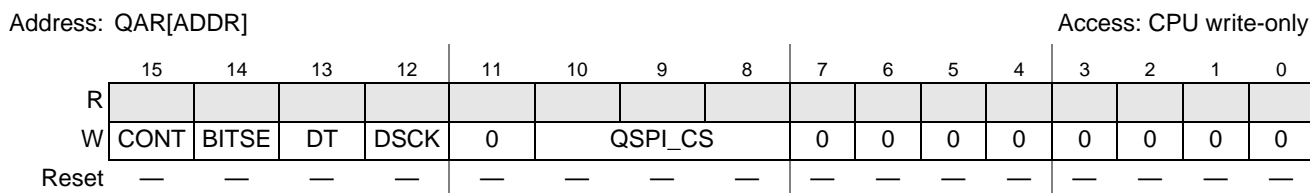
Figure 30-9. Command RAM Registers (QCR0–QCR15)

Table 30-9. QCR0–QCR15 Field Descriptions

Field	Description
15 CONT	Continuous. 0 Chip selects return to inactive level defined by QWR[CSIV] when a single word transfer is complete. 1 Chip selects return to inactive level defined by QWR[CSIV] only after the transfer of the queue entries (max of 16 words). **Note:** To keep the chip selects asserted for transfers beyond 16 words, the QWR[CSIV] bit must be set to control the level that the chip selects return to after the first transfer.
14 BITSE	Bits per transfer enable. 0 Eight bits 1 Number of bits set in QMR[BITS]
13 DT	Delay after transfer enable. 0 Default reset value. 1 The QSPI provides a variable delay at the end of serial transfer to facilitate interfacing with peripherals that have a latency requirement. The delay between transfers is determined by QDLYR[DTL].
12 DSCK	Chip select to QSPI_CLK delay enable. 0 Chip select valid to QSPI_CLK transition is one-half QSPI_CLK period. 1 QDLYR[QCD] specifies the delay from QSPI_CS valid to QSPI_CLK.
11	Reserved, must be cleared.
10–8 QSPI_CS	Peripheral chip selects. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 10-8 map directly to the corresponding QSPI_CS*n* pins. If more than three chip selects are needed, then an external demultiplexor can be used with the QSPI_CS*n* pins. **Note:** Not all chip selects may be available on all device packages. See Chapter 2, "Signal Descriptions," for details on which chip selects are pinned-out.
7–0	Reserved, must be cleared.

30.4 Functional Description

The QSPI uses a dedicated 80-byte block of static RAM accessible to the module and CPU to perform
queued operations. The RAM is divided into three segments:

- 16 command control bytes (command RAM)
- 32 transmit data bytes (transmit data RAM)

- 32 receive data bytes (receive data RAM)

The RAM is organized so that 1 byte of command control data, 1 word of transmit data, and 1 word of receive data comprise 1 of the 16 queue entries (0x0–0xF).

NOTE

Throughout ColdFire documentation, the term word is used to designate a 16-bit data unit. The only exceptions to this appear in discussions of serial communication modules such as QSPI that support variable-length data units. To simplify these discussions, the functional unit is referred to as a word regardless of length.

The user initiates QSPI operation by loading a queue of commands in command RAM, writing transmit data into transmit RAM, and then enabling the QSPI data transfer. The QSPI executes the queued commands and sets the completion flag in the QSPI interrupt register (QIR[SPIF]) to signal their completion. As another option, QIR[SPIFE] can be enabled to generate an interrupt.

The QSPI uses four queue pointers. The user can access three of them through fields in QSPI wrap register (QWR):

- New queue pointer (QWR[NEWQP])—points to the first command in the queue
- Internal queue pointer—points to the command currently being executed
- Completed queue pointer (QWR[CPTQP])—points to the last command executed
- End queue pointer (QWR[ENDQP]) —points to the final command in the queue

The internal pointer is initialized to the same value as QWR[NEWQP]. During normal operation, the following sequence repeats:

1. The command pointed to by the internal pointer is executed.
2. The value in the internal pointer is copied into QWR[CPTQP].
3. The internal pointer is incremented.

Execution continues at the internal pointer address unless the QWR[NEWQP] value is changed. After each command is executed, QWR[ENDQP] and QWR[CPTQP] are compared. When a match occurs, QIR[SPIF] is set and the QSPI stops unless wraparound mode is enabled. Setting QWR[WREN] enables wraparound mode.

QWR[NEWQP] is cleared at reset. When the QSPI is enabled, execution begins at address 0x0 unless another value has been written into QWR[NEWQP]. QWR[ENDQP] is cleared at reset but is changed to show the last queue entry before the QSPI is enabled. QWR[NEWQP] and QWR[ENDQP] can be written at any time. When the QWR[NEWQP] value changes, the internal pointer value also changes unless a transfer is in progress, in which case the transfer completes normally. Leaving QWR[NEWQP] and QWR[ENDQP] set to 0x0 causes a single transfer to occur when the QSPI is enabled.

Data is transferred relative to QSPI_CLK, which can be generated in any one of four combinations of phase and polarity using QMR[CPHA,CPOL]. Data is transferred with the most significant bit (msb) first. The number of bits transferred defaults to 8, but can be set to any value between 8 and 16 by writing a value into the BITSE field of the command RAM (QCR[BITSE]).

30.4.1   QSPI RAM

The QSPI contains an 80-byte block of static RAM that can be accessed by the user and the QSPI. This RAM does not appear in the device memory map, because it can only be accessed by the user indirectly through the QSPI address register (QAR) and the QSPI data register (QDR). The RAM is divided into three segments with 16 addresses each:

- Receive data RAM—the initial destination for all incoming data
- Transmit data RAM—a buffer for all out-bound data
- Command RAM—where commands are loaded

The transmit data and command RAM are user write-only. The receive RAM is user read-only. Figure 30-10 shows the RAM configuration. The RAM contents are undefined immediately after a reset.

The command and data RAM in the QSPI are indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data, and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR] and causes the value in QAR to increment. Correspondingly, a read from QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

Relative Address	Register	Function
0x00	QTR0	Transmit RAM
0x01	QTR1	
...	...	16 bits wide
0x0F	QTR15	

0x10	QRR0	Receive RAM
0x11	QRR1	
...	...	16 bits wide
0x1F	QRR15	

0x20	QCR0	Command RAM
0x21	QCR1	
...	...	8 bits wide
0x2F	QCR15	

Figure 30-10. QSPI RAM Model

30.4.1.1   Receive RAM

Data received by the QSPI is stored in the receive RAM segment located at 0x10 to 0x1F in the QSPI RAM space. Read this segment to retrieve data from the QSPI. Data words with less than 16 bits are stored in

the least significant bits of the RAM. Unused bits in a receive queue entry are set to zero upon completion of the individual queue entry. Receive RAM is not writeable.

QWR[CPTQP] shows which queue entries have been executed. The user can query this field to determine which locations in receive RAM contain valid data.

30.4.1.2 Transmit RAM

Data to be transmitted by the QSPI is stored in the transmit RAM segment located at addresses 0x0 to 0xF. The user normally writes 1 word into this segment for each queue command to be executed. The user cannot read data in the transmit RAM.

Outbound data must be written to transmit RAM in a right-justified format. The unused bits are ignored. The QSPI copies the data to its data serializer (shift register) for transmission. The data is transmitted most significant bit first and remains in transmit RAM until overwritten by the user.

30.4.1.3 Command RAM

The CPU writes one byte of control information to this segment for each QSPI command to be executed. Command RAM, referred to as QCR0–15, is write-only memory from a user's perspective.

Command RAM consists of 16 bytes, each divided into two fields. The peripheral chip select field controls the QSPI_CS signal levels for the transfer. The command control field provides transfer options.

A maximum of 16 commands can be in the queue. Queue execution proceeds from the address in QWR[NEWQP] through the address in QWR[ENDQP].

The QSPI executes a queue of commands defined by the control bits in each command RAM entry that sequence the following actions:

- Chip-select pins are activated.
- Data is transmitted from the transmit RAM and received into the receive RAM.
- The synchronous transfer clock QSPI_CLK is generated.

Before any data transfers begin, control data must be written to the command RAM, and any out-bound data must be written to the transmit RAM. Also, the queue pointers must be initialized to the first and last entries in the command queue.

Data transfer is synchronized with the internally generated QSPI_CLK, whose phase and polarity are controlled by QMR[CPHA] and QMR[CPOL]. These control bits determine which QSPI_CLK edge is used to drive outgoing data and to latch incoming data.

30.4.2 Baud Rate Selection

The maximum QSPI clock frequency is one-fourth the clock frequency of the internal bus clock ($f_{sys/3}$). Baud rate is selected by writing a value from 2–255 into QMR[BAUD]. The QSPI uses a prescaler to derive the QSPI_CLK rate from the internal bus clock divided by two. Table 30-10 shows the QSPI_CLK frequency as a function of internal bus clock and baud rate.

A baud rate value of zero turns off the QSPI_CLK.

The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression:

$$QMR[BAUD] = \frac{f_{sys/3}}{2 \times [\text{desired QSPI\_CLK baud rate}]}$$ **Eqn. 30-1**

Table 30-10. QSPI_CLK Frequency as Function of Internal Bus Clock and Baud Rate

Internal Bus Clock = 80 MHz	
QMR [BAUD]	QSPI_CLK
2	20 MHz
4	10 MHz
8	5 MHz
16	2.5 MHz
32	1.25 MHz
255	156.9 kHz

30.4.3 Transfer Delays

The QSPI supports programmable delays for the QSPI_CS signals before and after a transfer. The time between QSPI_CS assertion and the leading QSPI_CLK edge, and the time between the end of one transfer and the beginning of the next, are both independently programmable.

The chip select to clock delay enable bit in the command RAM, QCR[DSCK], enables the programmable delay period from QSPI_CS assertion until the leading edge of QSPI_CLK. QDLYR[QCD] determines the period of delay before the leading edge of QSPI_CLK. The following expression determines the actual delay before the QSPI_CLK leading edge:

$$\text{QSPI\_CS-to-QSPI\_CLK delay} = \frac{QDLYR[QCD]}{f_{sys/3}}$$ **Eqn. 30-2**

QDLYR[QCD] has a range of 1–127.

When QDLYR[QCD] or QCR[DSCK] equals zero, the standard delay of one-half the QSPI_CLK period is used.

The command RAM delay after transmit enable bit, QCR[DT], enables the programmable delay period from the negation of the QSPI_CS signals until the start of the next transfer. The delay after transfer can be used to provide a peripheral deselect interval. A delay can also be inserted between consecutive transfers to allow serial A/D converters to complete conversion. There are two transfer delay options: the user can choose to delay a standard period after serial transfer is complete or can specify a delay period. Writing a value to QDLYR[DTL] specifies a delay period. QCR[DT] determines whether the standard delay period (DT = 0) or the specified delay period (DT = 1) is used. The following expression is used to calculate the delay when DT equals 1:

$$\text{Delay after transfer} = \frac{32 \times QDLYR[DTL]}{f_{sys/3}} \quad (DT = 1)$$ **Eqn. 30-3**

where QDLYR[DTL] has a range of 1–255. A zero value for DTL causes a delay-after-transfer value of $8192/f_{sys/3}$. Standard delay period (DT = 0) is calculated by the following:

$$\text{Standard delay after transfer} = \frac{17}{f_{sys/3}} \quad (DT = 0)$$

Eqn. 30-4

Adequate delay between transfers must be specified for long data streams because the QSPI module requires time to load a transmit RAM entry for transfer. Receiving devices need at least the standard delay between successive transfers. If the internal bus clock is operating at a slower rate, the delay between transfers must be increased proportionately.

30.4.4 Transfer Length

There are two transfer length options. The user can choose a default value of 8 bits or a programmed value of 8 to 16 bits. The programmed value must be written into QMR[BITS]. The command RAM bits per transfer enable field, QCR[BITSE], determines whether the default value (BITSE = 0) or the BITS[3–0] value (BITSE = 1) is used. QMR[BITS] indicates the required number of bits to be transferred, with the default value of 16 bits.

30.4.5 Data Transfer

The transfer operation is initiated by setting QDLYR[SPE]. Shortly after QDLYR[SPE] is set, the QSPI executes the command at the command RAM address pointed to by QWR[NEWQP]. Data at the pointer address in transmit RAM is loaded into the data serializer and transmitted. Data that is simultaneously received is stored at the pointer address in receive RAM.

When the proper number of bits has been transferred, the QSPI stores the working queue pointer value in QWR[CPTQP], increments the working queue pointer, and loads the next data for transfer from the transmit RAM. The command pointed to by the incremented working queue pointer is executed next unless a new value has been written to QWR[NEWQP]. If a new queue pointer value is written while a transfer is in progress, the current transfer is completed normally.

When the CONT bit in the command RAM is set, the QSPI_CS*n* signals are asserted between transfers. When CONT is cleared, QSPI_CS*n* are negated between transfers. The QSPI_CS*n* signals are not high impedance.

When the QSPI reaches the end of the queue, it asserts the SPIF flag, QIR[SPIF]. If QIR[SPIFE] is set, an interrupt request is generated when QIR[SPIF] is asserted. Then the QSPI clears QDLYR[SPE] and stops, unless wraparound mode is enabled.

Wraparound mode is enabled by setting QWR[WREN]. The queue can wrap to pointer address 0x0, or to the address specified by QWR[NEWQP], depending on the state of QWR[WRTO].

In wraparound mode, the QSPI cycles through the queue continuously, even while requesting interrupt service. QDLYR[SPE] is not cleared when the last command in the queue is executed. New receive data overwrites previously received data in the receive RAM. Each time the end of the queue is reached,

QIR[SPIFE] is set. QIR[SPIF] is not automatically reset. If interrupt driven QSPI service is used, the service routine must clear QIR[SPIF] to abort the current request. Additional interrupt requests during servicing can be prevented by clearing QIR[SPIFE].

There are two recommended methods of exiting wraparound mode: clearing QWR[WREN] or setting QWR[HALT]. Exiting wraparound mode by clearing QDLYR[SPE] is not recommended because this may abort a serial transfer in progress. The QSPI sets SPIF, clears QDLYR[SPE], and stops the first time it reaches the end of the queue after QWR[WREN] is cleared. After QWR[HALT] is set, the QSPI finishes the current transfer, then stops executing commands. After the QSPI stops, QDLYR[SPE] can be cleared.

30.5   Initialization/Application Information

The following steps are necessary to set up the QSPI 12-bit data transfers and a QSPI_CLK of 5 MHz. The QSPI RAM is set up for a queue of 12 transfers. All three QSPI_CS signals are used in this example.

1. Write the QMR with 0xB308 to set up 12-bit data words with the data shifted on the falling clock edge, and a QSPI_CLK frequency of 5 MHz (assuming a 80-MHz internal bus clock).
2. Write QDLYR with the desired delays.
3. Write QIR with 0xD00F to enable write collision, abort bus errors, and clear any interrupts.
4. Write QAR with 0x0020 to select the first command RAM entry.
5. Write QDR with 0x7E00, 0x7E00, 0x7E00, 0x7E00, 0x7D00, 0x7D00, 0x7D00, 0x7D00, 0x7B00, 0x7B00, 0x7B00, and 0x7B00 to set up four transfers for each chip select. The chip selects are active low in this example.
6. Write QAR with 0x0000 to select the first transmit RAM entry.
7. Write QDR with twelve 12-bit words of data.
8. Write QWR with 0x0B00 to set up a queue beginning at entry 0 and ending at entry 11.
9. Set QDLYR[SPE] to enable the transfers.
10. Wait until the transfers are complete. QIR[SPIF] is set when the transfers are complete.
11. Write QAR with 0x0010 to select the first receive RAM entry.
12. Read QDR to get the received data for each transfer.
13. Repeat steps 5 through 13 to do another transfer.

Chapter 31
UART Modules

31.1   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.

31.1.1   Overview

The internal bus clock can clock each of the three independent UARTs, eliminating the need for an external UART clock. As Figure 31-1 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
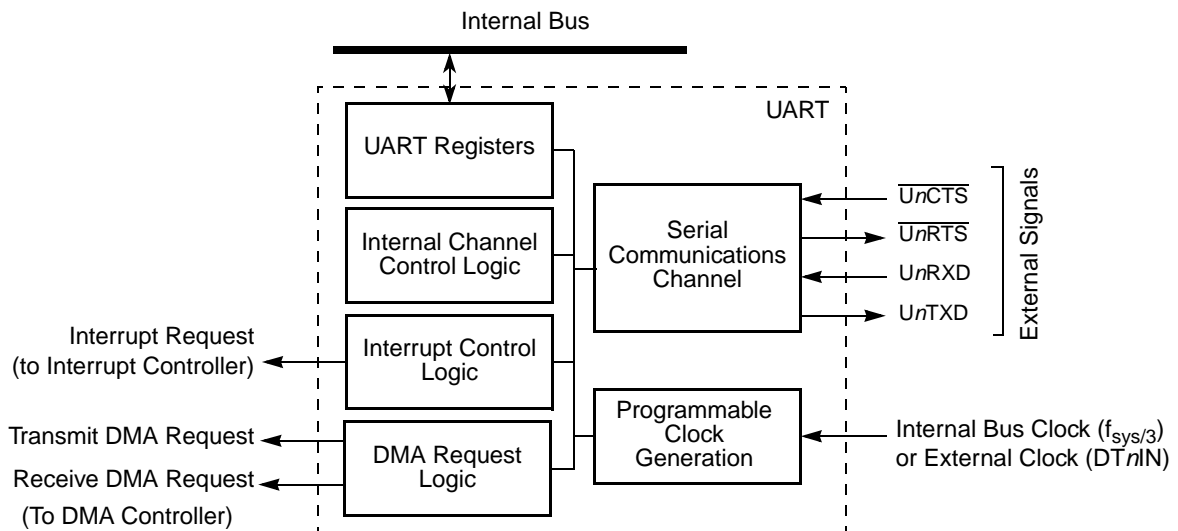
Figure 31-1. UART Block Diagram

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 31.4.2.1, "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 31.4.2.2, "Receiver."

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 13, "General Purpose I/O Module") prior to configuring the UART module.

31.1.2   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

31.2 External Signal Description

Table 31-1 briefly describes the UART module signals.

Table 31-1. 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.
U*n*CTS	Clear-to- Send. This input can generate an interrupt on a change of state.
U*n*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 U*n*CTS, U*n*RTS can control serial data flow.

Figure 31-2 shows a signal configuration for a UART/RS-232 interface.
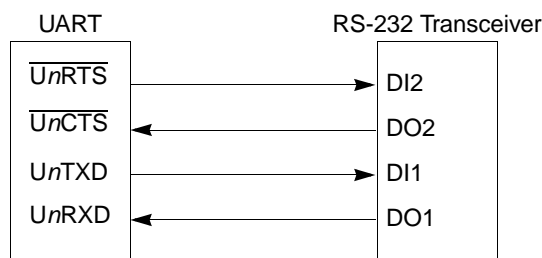
Figure 31-2. UART/RS-232 Interface

31.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in Section 31.5, "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 31-2. UART Module Memory Map

Address UART0 UART1 UART2	Register	Width (bit)	Access	Reset Value	Section/Page
0xFC06_0000 0xFC06_4000 0xFC06_8000	UART Mode Registers[1] (UMR1n), (UMR2n)	8	R/W	0x00	31.3.1/31-5 31.3.2/31-6
0xFC06_0004 0xFC06_4004 0xFC06_8004	UART Status Register (USRn)	8	R	0x00	31.3.3/31-8
	UART Clock Select Register[1] (UCSRn)	8	W	See Section	31.3.4/31-9
0xFC06_0008 0xFC06_4008 0xFC06_8008	UART Command Registers (UCRn)	8	W	0x00	31.3.5/31-9
0xFC06_000C 0xFC06_400C 0xFC06_800C	UART Receive Buffers (URBn)	8	R	0xFF	31.3.6/31-11
	UART Transmit Buffers (UTBn)	8	W	0x00	31.3.7/31-12
0xFC06_0010 0xFC06_4010 0xFC06_8010	UART Input Port Change Register (UIPCRn)	8	R	See Section	31.3.8/31-12
	UART Auxiliary Control Register (UACRn)	8	W	0x00	31.3.9/31-13
0xFC06_0014 0xFC06_4014 0xFC06_8014	UART Interrupt Status Register (UISRn)	8	R	0x00	31.3.10/31-13
	UART Interrupt Mask Register (UIMRn)	8	W	0x00	
0xFC06_0018 0xFC06_4018 0xFC06_8018	UART Baud Rate Generator Register (UBG1n)	8	W[2]	0x00	31.3.11/31-15
0xFC06_001C 0xFC06_401C 0xFC06_801C	UART Baud Rate Generator Register (UBG2n)	8	W[2]	0x00	31.3.11/31-15
0xFC06_0034 0xFC06_4034 0xFC06_8034	UART Input Port Register (UIPn)	8	R	0xFF	31.3.12/31-15
0xFC06_0038 0xFC06_4038 0xFC06_8038	UART Output Port Bit Set Command Register (UOP1n)	8	W[2]	0x00	31.3.13/31-16
0xFC06_003C 0xFC06_403C 0xFC06_803C	UART Output Port Bit Reset Command Register (UOP0n)	8	W[2]	0x00	31.3.13/31-16

[1] UMR1n, UMR2n, and UCSRn 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.

31.3.1 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)   Access: User read/write[1]
         0xFC06_4000 (UMR11)
         0xFC06_8000 (UMR12)

	7	6	5	4	3	2	1	0
R	RXRTS	RXIRQ/ FFULL	ERR	PM	PM	PT	B/C	B/C
W								
Reset:	0	0	0	0	0	0	0	0
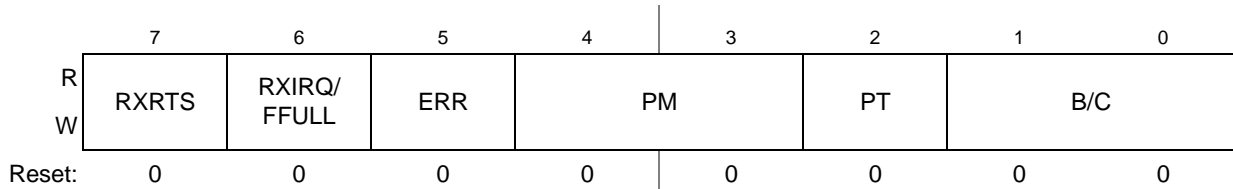
[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 31-3. UART Mode Registers 1 (UMR1*n*)

Table 31-3. UMR1*n* Field Descriptions

Field	Description
7 RXRTS	Receiver request-to-send. Allows the $\overline{Un\text{RTS}}$ output to control the $\overline{Un\text{CTS}}$ input of the transmitting device to prevent receiver overrun. If the receiver and transmitter are incorrectly programmed for $\overline{Un\text{RTS}}$ control, $\overline{Un\text{RTS}}$ control is disabled for both. Transmitter RTS control is configured in UMR2*n*[TXRTS]. 0 The receiver has no effect on $\overline{Un\text{RTS}}$. 1 When a valid start bit is received, $\overline{Un\text{RTS}}$ is negated if the UART's FIFO is full. $\overline{Un\text{RTS}}$ 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 31.3.5, "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 31-3. UMR1*n* Field Descriptions (continued)

Field	Description
2 PT	Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11). | PM	Parity Mode	Parity Type (PT= 0)	Parity Type (PT= 1)
---	---	---	---
00	With parity	Even parity	Odd parity
01	Force parity	Low parity	High parity
10	No parity	N/A	
11	Multidrop mode	Data character	Address character	
1–0 B/C	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

31.3.2   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)   Access: User read/write[1]
         0xFC06_4000 (UMR21)
         0xFC06_8000 (UMR22)

	7	6	5	4	3	2	1	0
R	CM		TXRTS	TXCTS	SB			
W								
Reset:	0	0	0	0	0	0	0	0
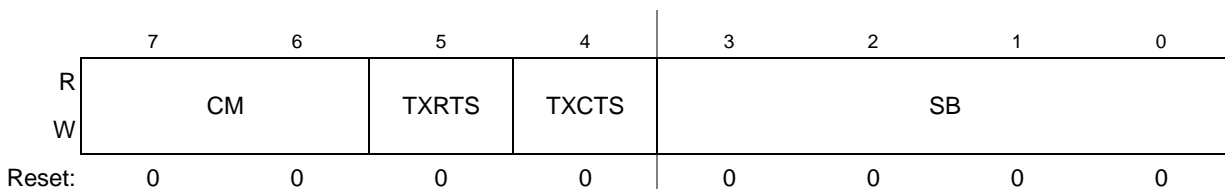
[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 31-4. UART Mode Registers 2 (UMR2*n*)

Table 31-4. UMR2*n* Field Descriptions

Field	Description
7–6 CM	Channel mode. Selects a channel mode. Section 31.4.3, "Looping Modes," describes individual modes. 00 Normal 01 Automatic echo 10 Local loopback 11 Remote loopback
5 TXRTS	Transmitter ready-to-send. Controls negation of U*n*RTS to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same UART for U*n*RTS control is not permitted and disables U*n*RTS control for both. 0 The transmitter has no effect on U*n*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.
4 TXCTS	Transmitter clear-to-send. If TXCTS and TXRTS are set, TXCTS controls the operation of the transmitter. 0 U*n*CTS has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of U*n*CTS each time it is ready to send a character. If U*n*CTS is asserted, the character is sent; if it is deasserted, the signal U*n*TXD remains in the high state and transmission is delayed until U*n*CTS is asserted. Changes in U*n*CTS as a character is being sent do not affect its transmission.
3–0 SB	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>

31.3.3 UART Status Registers (USR*n*)

The USR*n* registers show the status of the transmitter, the receiver, and the FIFO.

Address: 0xFC06_0004 (USR0)   Access: User read-only
              0xFC06_4004 (USR1)
              0xFC06_8004 (USR2)

	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
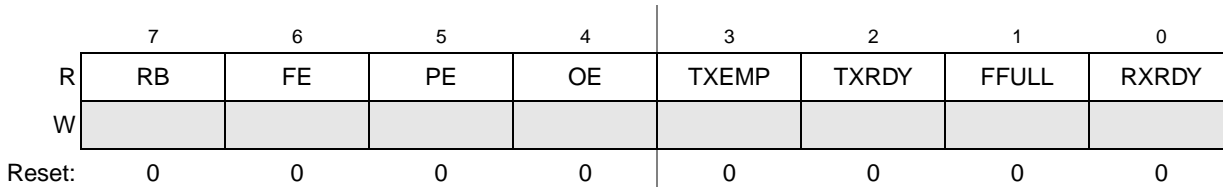
Figure 31-5. UART Status Registers (USR*n*)

Table 31-5. 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 TEMP	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 31-5. 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.

31.3.4   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 31.4.1, "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)   Access: User write-only
         0xFC06_4004 (UCSR1)
         0xFC06_8004 (UCSR2)

	7	6	5	4	3	2	1	0
R								
W			RCS				TCS	
Reset:			See Note				See Note	
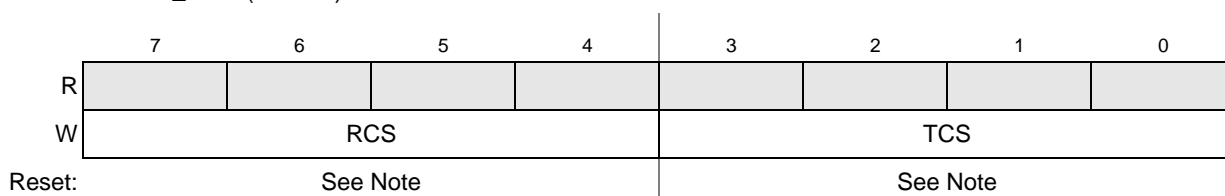
Note: The RCS and TCS reset values are set so the receiver and transmiter use the prescaled internal bus clock as their clock source.

Figure 31-6. UART Clock Select Registers (UCSR*n*)

Table 31-6. UCSR*n* Field Descriptions

Field	Description
7–4 RCS	Receiver clock select. Selects the clock source for the receiver. 1101 Prescaled internal bus clock ($f_{sys/3}$) 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 ($f_{sys/3}$) 1110 DT*n*IN divided by 16 1111 DT*n*IN

31.3.5   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)   Access: User write-only
         0xFC06_4008 (UCR1)
         0xFC06_8008 (UCR2)

	7	6	5	4	3	2	1	0
R								
W	0		MISC		TC		RC	
Reset:	0	0	0	0	0	0	0	0
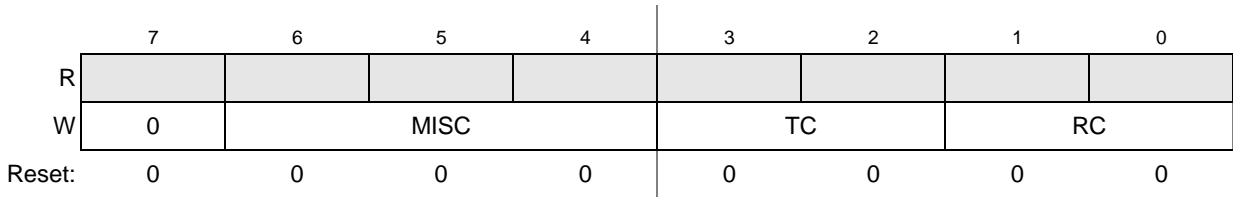
Figure 31-7. UART Command Registers (UCR*n*)

Table 31-7 describes UCR*n* fields and commands. Examples in Section 31.4.2, "Transmitter and Receiver Operating Modes," show how these commands are used.

Table 31-7. UCR*n* Field Descriptions

Field	Description
7	Reserved, must be cleared.
6–4 MISC	MISC Field (this field selects a single command) |	Command	Description | |---|---|---| | 000 | NO COMMAND | — | | 001 | RESET MODE REGISTER POINTER | Causes the mode register pointer to point to UMR1*n*. | | 010 | RESET RECEIVER | Immediately disables the receiver, clears USR*n*[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. | | 011 | RESET TRANSMITTER | Immediately disables the transmitter and clears USR*n*[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. | | 100 | RESET ERROR STATUS | Clears USR*n*[RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received. | | 101 | RESET BREAK – CHANGE INTERRUPT | Clears the delta break bit, UISR*n*[DB]. | | 110 | START BREAK | Forces U*n*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 U*n*CTS. | | 111 | STOP BREAK | Causes U*n*TXD to go high (mark) within two bit times. Any characters in the transmit buffer are sent. |

Table 31-7. UCR*n* Field Descriptions (continued)

Field	Description
3–2 TC	Transmit command field. Selects a single transmit command.

	Command		Description
00	NO ACTION TAKEN		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.
01	TRANSMITTER ENABLE		Enables operation of the UART's transmitter. USR*n*[TXEMP,TXRDY] are set. If the transmitter is already enabled, this command has no effect.
10	TRANSMITTER DISABLE		Terminates transmitter operation and clears USR*n*[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.
11	—		Reserved, do not use.

Field	Description
1–0 RC	Receive command field. Selects a single receive command.

	Command		Description
00	NO ACTION TAKEN		Causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.
01	RECEIVER ENABLE		If the UART module is not in multidrop mode (UMR1*n*[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.
10	RECEIVER DISABLE		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.
11	—		Reserved, do not use.

31.3.6 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 31-18). RB contains the character in the receiver.

Address: 0xFC06_000C (URB0)   Access: User read-only
         0xFC06_400C (URB1)
         0xFC06_800C (URB2)

	7	6	5	4	3	2	1	0
R				RB				
W								
Reset:	1	1	1	1	1	1	1	1
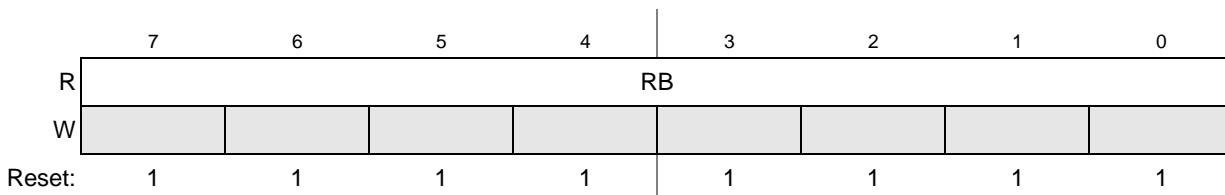
Figure 31-8. UART Receive Buffer (URB*n*)

31.3.7 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 31-9 shows UTB*n*. TB contains the character in the transmit buffer.

Address: 0xFC06_000C (UTB0)   Access: User write-only
         0xFC06_400C (UTB1)
         0xFC06_800C (UTB2)

	7	6	5	4	3	2	1	0
R								
W				TB				
Reset:	0	0	0	0	0	0	0	0
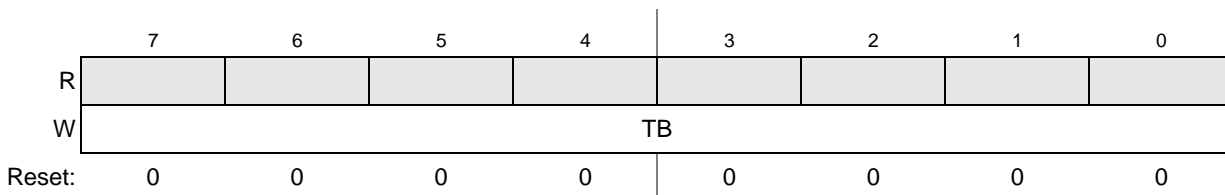
Figure 31-9. UART Transmit Buffer (UTB*n*)

31.3.8 UART Input Port Change Registers (UIPCR*n*)

The UIPCRs hold the current state and the change-of-state for $\overline{Un\text{CTS}}$.

Address: 0xFC06_0010 (UIPCR0)   Access: User read-only
         0xFC06_4010 (UIPCR1)
         0xFC06_8010 (UIPCR2)

	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	$\overline{Un\text{CTS}}$
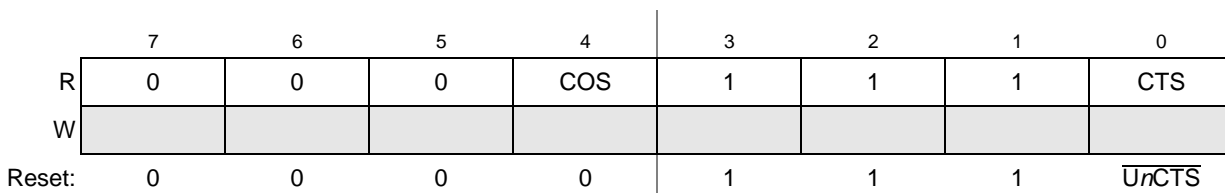
Figure 31-10. UART Input Port Changed Registers (UIPCR*n*)

Table 31-8. 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 $\overline{UnCTS}$ 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 $\overline{UnCTS}$. If $\overline{UnCTS}$ 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 $\overline{UnCTS}$ input is asserted. 1 The current state of the $\overline{UnCTS}$ input is deasserted.

31.3.9 UART Auxiliary Control Register (UACR*n*)

The UACRs control the input enable.

Address: 0xFC06_0010 (UACR0)   Access: User write-only
            0xFC06_4010 (UACR1)
            0xFC06_8010 (UACR2)

	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 31-11. UART Auxiliary Control Registers (UACR*n*)

Table 31-9. 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 $\overline{UnCTS}$ input (if UIMR*n*[COS] = 1).

31.3.10 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 UISRn regardless of UIMRn settings. UISRn
is cleared when the UART module is reset.

Address: 0xFC06_0014 (UISR0)   Access: User read/write
         0xFC06_4014 (UISR1)
         0xFC06_8014 (UISR2)

	7	6	5	4	3	2	1	0
R (UISRn)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
W (UIMRn)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
Reset:	0	0	0	0	0	0	0	0

Figure 31-12. UART Interrupt Status/Mask Registers (UISRn/UIMRn)

Table 31-10. UISRn/UIMRn Field Descriptions

Field	Description
7 COS	Change-of-state. 0 UIPCRn[COS] is not selected. 1 Change-of-state occurred on $\overline{Un\text{CTS}}$ and was programmed in UACRn[IEC] to cause an interrupt.
6–3	Reserved, must be cleared.
2 DB	Delta break. 0 No new break-change condition to report. Section 31.3.5, "UART Command Registers (UCRn)," describes the RESET BREAK-CHANGE INTERRUPT command. 1 The receiver detected the beginning or end of a received break.
1 FFULL/ RXRDY	Status of FIFO or receiver, depending on UMR1[FFULL/RXRDY] bit. Duplicate of USRn[FIFO] and USRn[RXRDY] <table><tr><td rowspan="2">UIMRn [FFULL/RXRDY]</td><td rowspan="2">UISRn [FFULL/RXRDY]</td><td colspan="2">UMR1n[FFULL/RXRDY]</td></tr><tr><td>0 (RXRDY)</td><td>1 (FIFO)</td></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>
0 TXRDY	Transmitter ready. This bit is the duplication of USRn[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.

31.3.11 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 31.4.1.2.1, "Internal Bus Clock Baud Rates."

Address: 0xFC06_0018 (UBG10)   Access: User write-only
        0xFC06_4018 (UBG11)
        0xFC06_8018 (UBG12)

	7	6	5	4	3	2	1	0
R								
W	Divider MSB							
Reset:	0	0	0	0	0	0	0	0
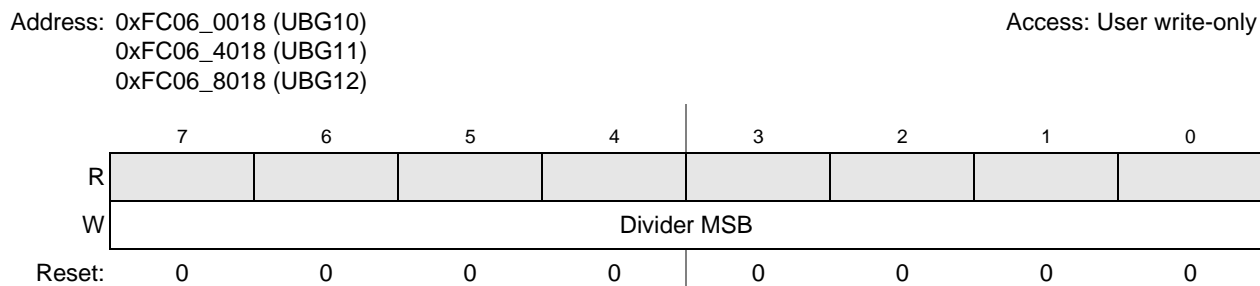
Figure 31-13. UART Baud Rate Generator Registers (UBG1*n*)

Address: 0xFC06_001C (UBG20)   Access: User write-only
        0xFC06_401C (UBG21)
        0xFC06_801C (UBG22)

	7	6	5	4	3	2	1	0
R								
W	Divider LSB							
Reset:	0	0	0	0	0	0	0	0
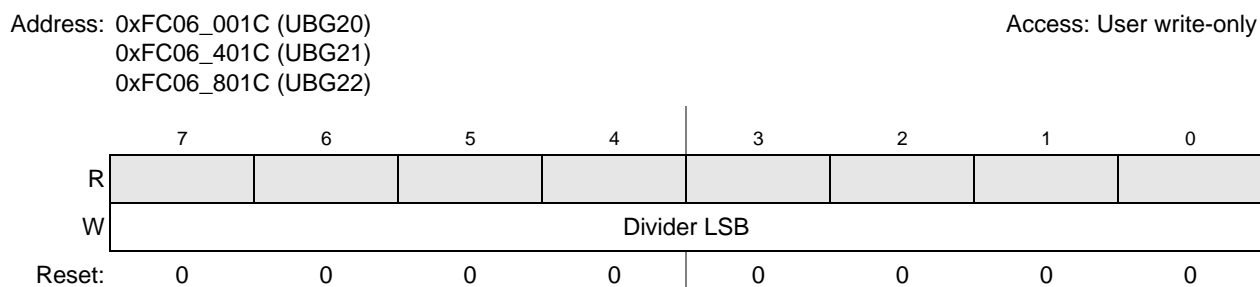
Figure 31-14. UART Baud Rate Generator Registers (UBG2*n*)

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.

31.3.12 UART Input Port Register (UIP*n*)

The UIP*n* registers show the current state of the $\overline{\text{U}n\text{CTS}}$ input.

Address: 0xFC06_0034 (UIP0)   Access: User read-only
        0xFC06_4034 (UIP1)
        0xFC06_8034 (UIP2)

	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
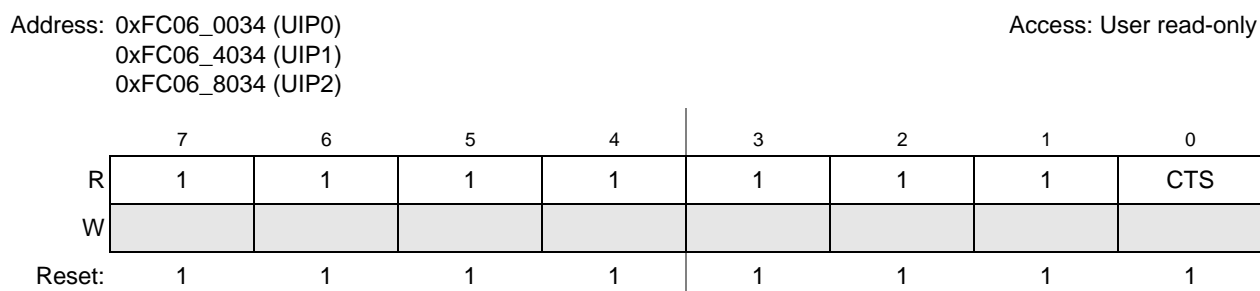
Figure 31-15. UART Input Port Registers (UIP*n*)

Table 31-11. UIP*n* Field Descriptions

Field	Description
7–1	Reserved
0 CTS	Current state of clear-to-send. The $\overline{UnCTS}$ 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 $\overline{UnCTS}$ input is logic 0. 1 The current state of the $\overline{UnCTS}$ input is logic 1.

31.3.13 UART Output Port Command Registers (UOP1*n*/UOP0*n*)

The $\overline{UnRTS}$ 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)   Access: User write-only
          0xFC06_003C (UOP00)
          0xFC06_4038 (UOP11)
          0xFC06_403C (UOP01)
          0xFC06_8038 (UOP12)
          0xFC06_803C (UOP02)

	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
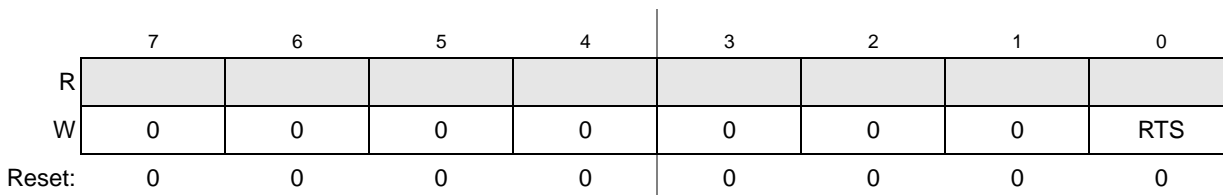
Figure 31-16. UART Output Port Command Registers (UOP1*n*/UOP0*n*)

Table 31-12. 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 $\overline{UnRTS}$ output. 0 Not affected. 1 Asserts $\overline{UnRTS}$ in UOP1. Negates $\overline{UnRTS}$ in UOP0.

31.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

31.4.1 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.

31.4.1.1 Programmable Divider

As Figure 31-17 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 31.3.11, "UART Baud Rate Generator Registers (UBG1n/UBG2n)."

The choice of DTIN or internal bus clock is programmed in the UCSR.
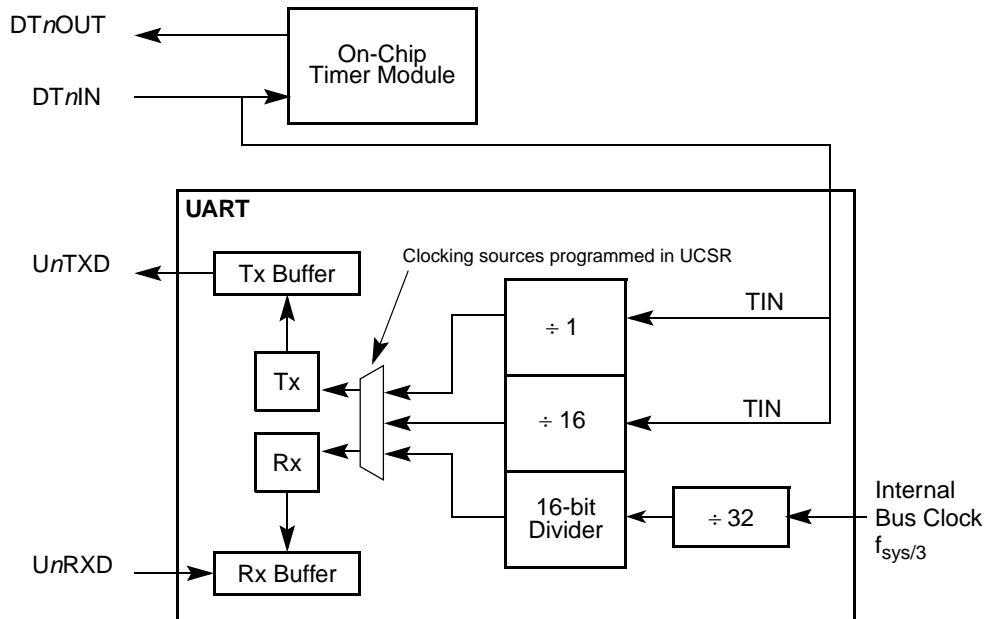
Figure 31-17. Clocking Source Diagram

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.

31.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

31.4.1.2.1 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:

$$\text{Baudrate} = \frac{f_{sys/3}}{[32 \times \text{divider}]}$$

Eqn. 31-1

Using a 80-MHz internal bus clock and letting baud rate equal 9600, then

$$\text{Divider} = \frac{80\text{MHz}}{[32 \times 9600]} = 260(\text{decimal}) = 0x0104(\text{hexadecimal}) \qquad \textit{Eqn. 31-2}$$

Therefore, UBG1*n* equals 0x01 and UBG2*n* equals 0x04.

31.4.1.2.2   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:

$$\text{Baudrate} = \frac{f_{extc}}{(16 \text{ or } 1)} \qquad \textit{Eqn. 31-3}$$

31.4.2   Transmitter and Receiver Operating Modes

Figure 31-18 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 31.3, "Memory Map/Register Definition."
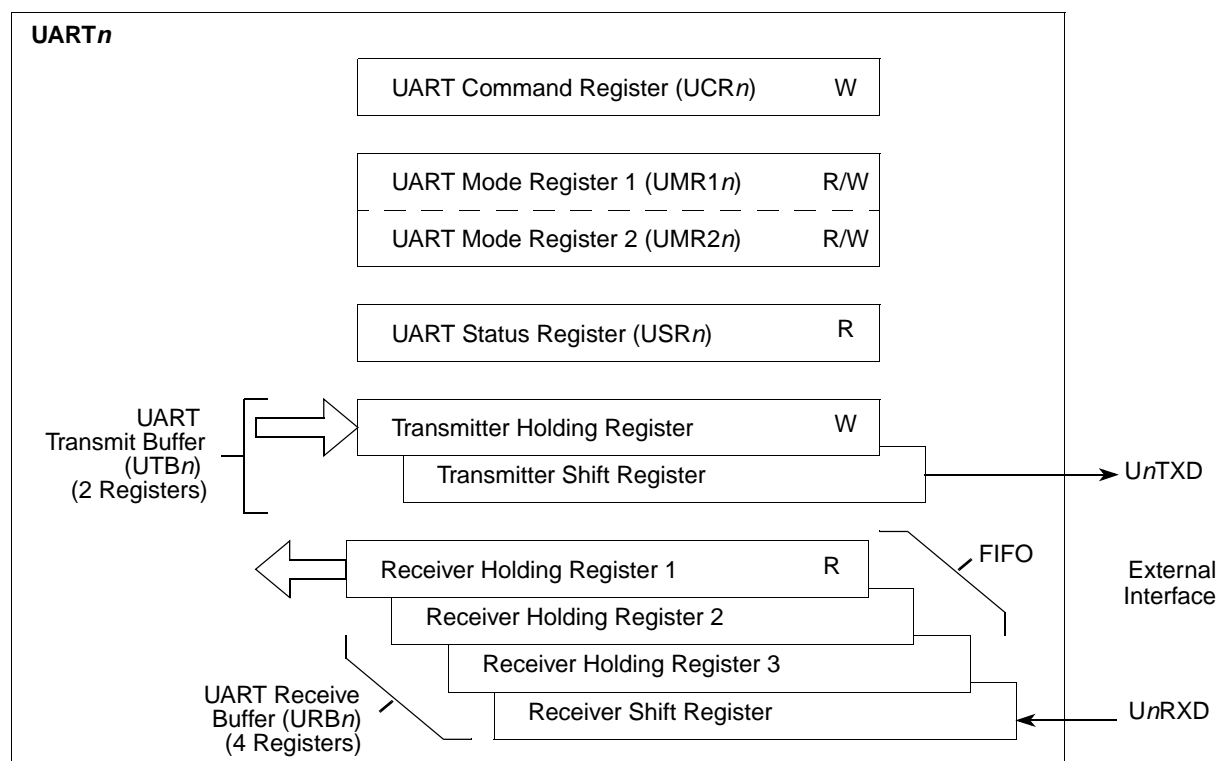
Figure 31-18. Transmitter and Receiver Functional Diagram

31.4.2.1   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 UnTXD output remains high (mark condition) and the transmitter empty bit (USRn[TXEMP]) is set. Transmission resumes and TXEMP is cleared when the CPU loads a new character into the UART transmit buffer (UTBn). 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 31.3.5, "UART Command Registers (UCRn)"). The transmitter is reenabled through the UCRn 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 UnTXD 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 31-19 shows the functional timing information for the transmitter.
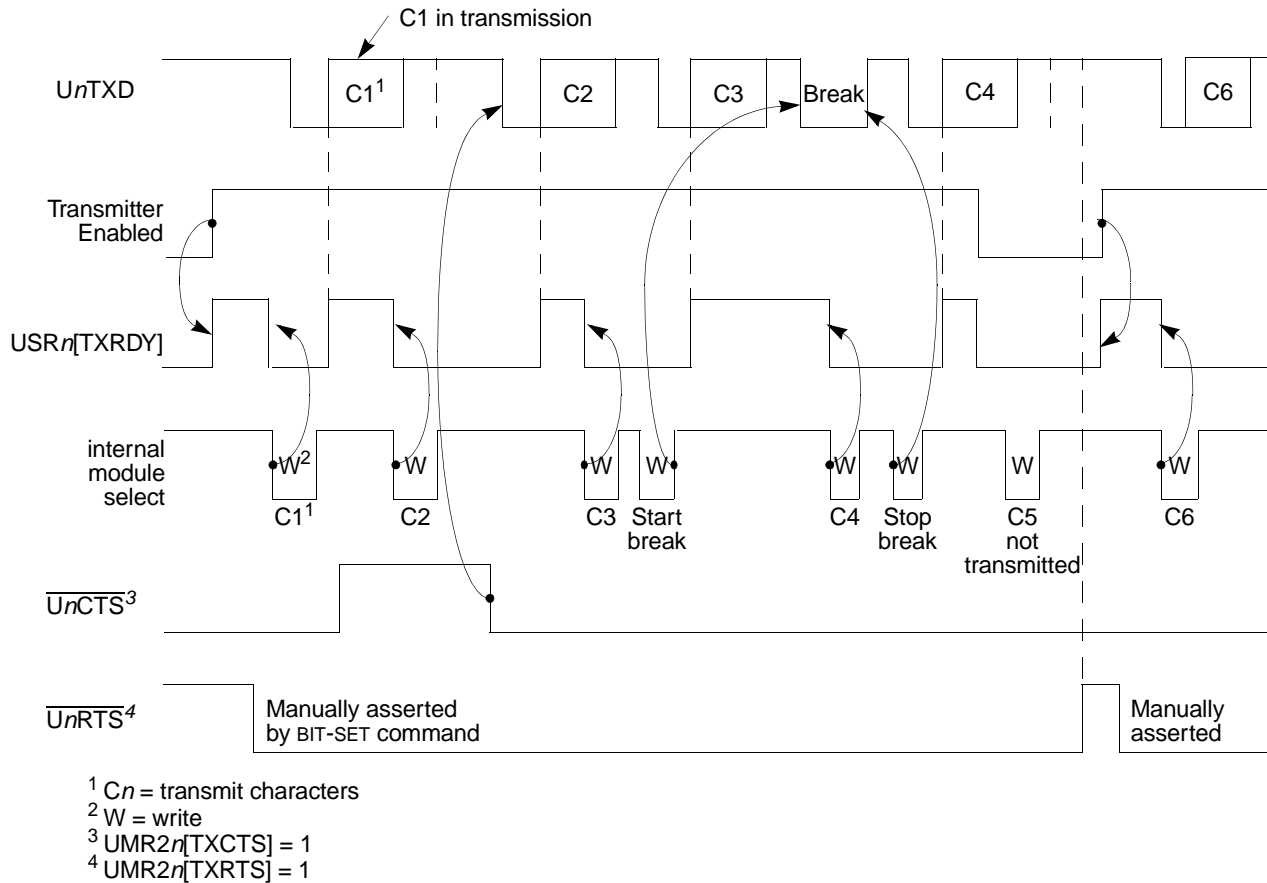
Figure 31-19. Transmitter Timing Diagram

[1] C*n* = transmit characters
[2] W = write
[3] UMR2*n*[TXCTS] = 1
[4] UMR2*n*[TXRTS] = 1

31.4.2.2 Receiver

The receiver is enabled through its UCR*n*, as described in Section 31.3.5, "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 USRn at the received character boundary. They are valid only if USRn[RXRDY] is set.

If a break condition is detected (UnRXD is low for the entire character including the stop bit), a character of all 0s loads into the receiver holding register and USRn[RB,RXRDY] are set. UnRXD 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. If the break begins in the middle of a character, receiver places the damaged character in the Rx FIFO and sets the corresponding USRn error bits and USRn[RXRDY]. Then, if the break lasts until the next character time, receiver places an all-zero character into the Rx FIFO and sets USRn[RB,RXRDY].

Figure 31-20 shows receiver functional timing.
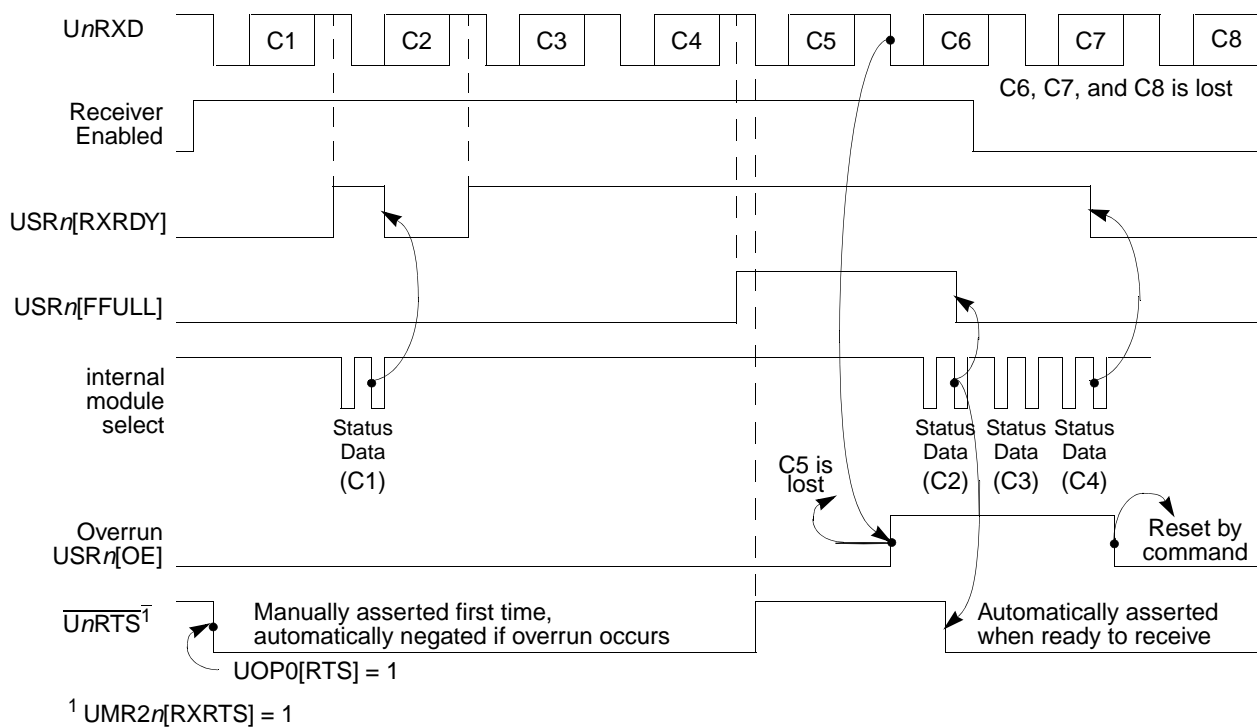
Figure 31-20. Receiver Timing Diagram

31.4.2.3   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 UnRXD (see Figure 31-18). 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 (UMR1n), status is provided in character or block modes.

USRn[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 UMR1n[ERR]:

- In character mode (UMR1n[ERR] = 0), status is given in the USRn for the character at the top of the FIFO.
- In block mode, the USRn 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 USRn does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The USRn 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 USRn[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.

31.4.3 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 31.3, "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.

31.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in Figure 31-21, 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.
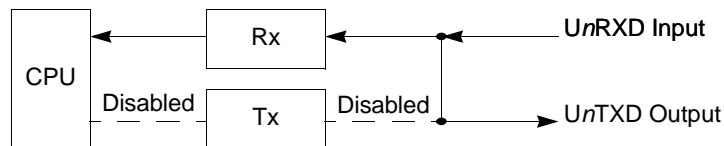
Figure 31-21. Automatic Echo

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.

31.4.3.2 Local Loopback Mode

Figure 31-22 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.
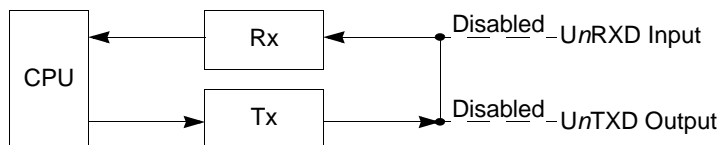
Figure 31-22. Local Loopback

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.

31.4.3.3 Remote Loopback Mode

In remote loopback mode, shown in Figure 31-23, 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.
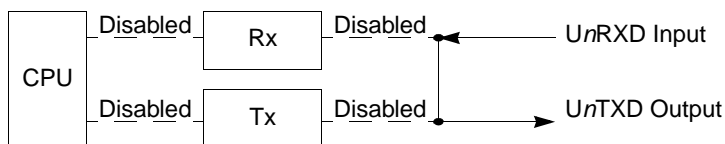
Figure 31-23. Remote Loopback

31.4.4 Multidrop Mode

Setting UMR1n[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 USRn[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 31-24.

Master Station

Peripheral Station
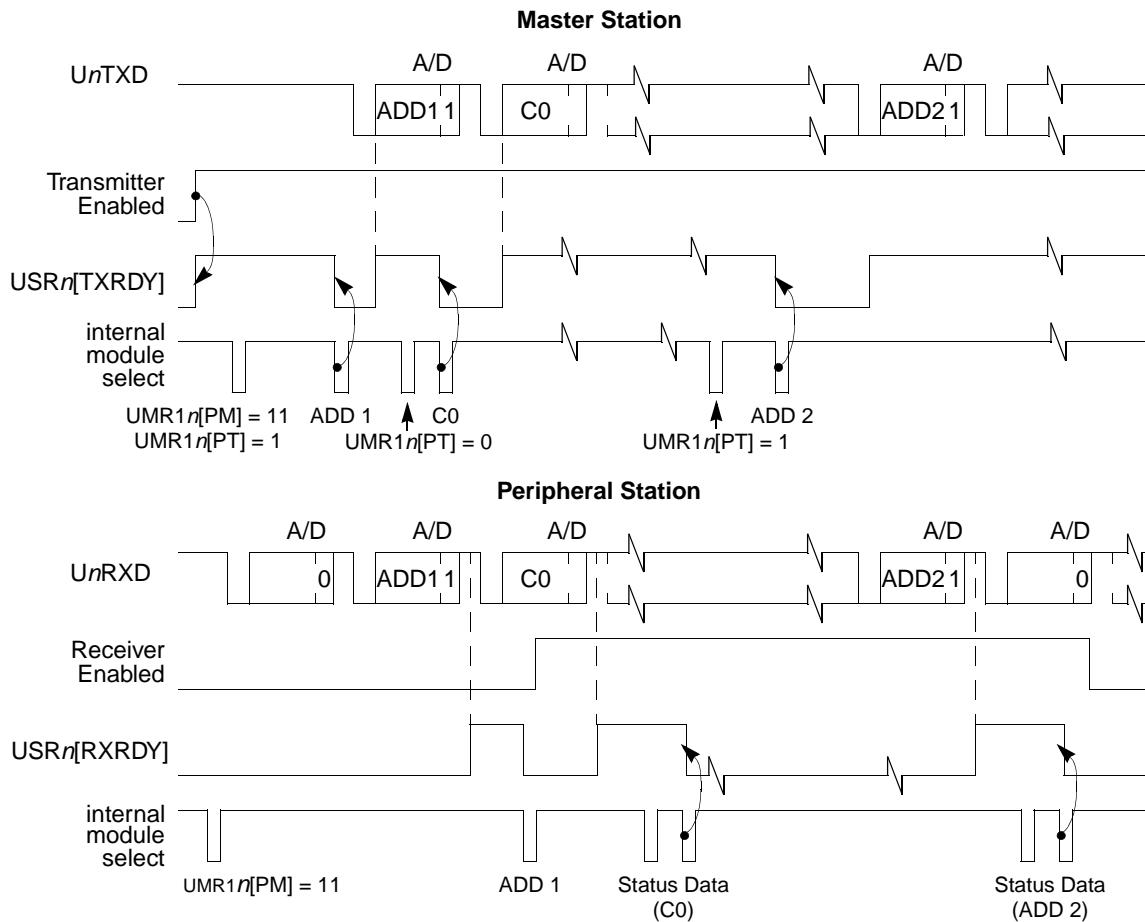
Figure 31-24. Multidrop Mode Timing Diagram

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 UMR1n[PT]. UMR1n 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 (USRn[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.

31.4.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

31.4.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

31.4.5.2 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.

31.5 Initialization/Application Information

The software flowchart, Figure 31-25, consists of:

- UART module initialization—These routines consist of SINIT and CHCHK (See Sheet 1 p. 31-29 and Sheet 2 p. 31-30). 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. 31-32 and Sheet 5 p. 31-33) 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. 31-32), 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.

31.5.1 Interrupt and DMA Request Initialization

31.5.1.1 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 14.2.9.1, "Interrupt Sources," for details on interrupt assignments for the UART modules.

1. Initialize the appropriate ICRx 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 31-13.

Table 31-13. 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

31.5.1.2   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 31-14 shows the DMA requests.

Table 31-14. UART DMA Requests

Register	Bit	DMA Request
UISR*n*	1	Receive DMA request
UISR*n*	0	Transmit DMA request

31.5.2 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.
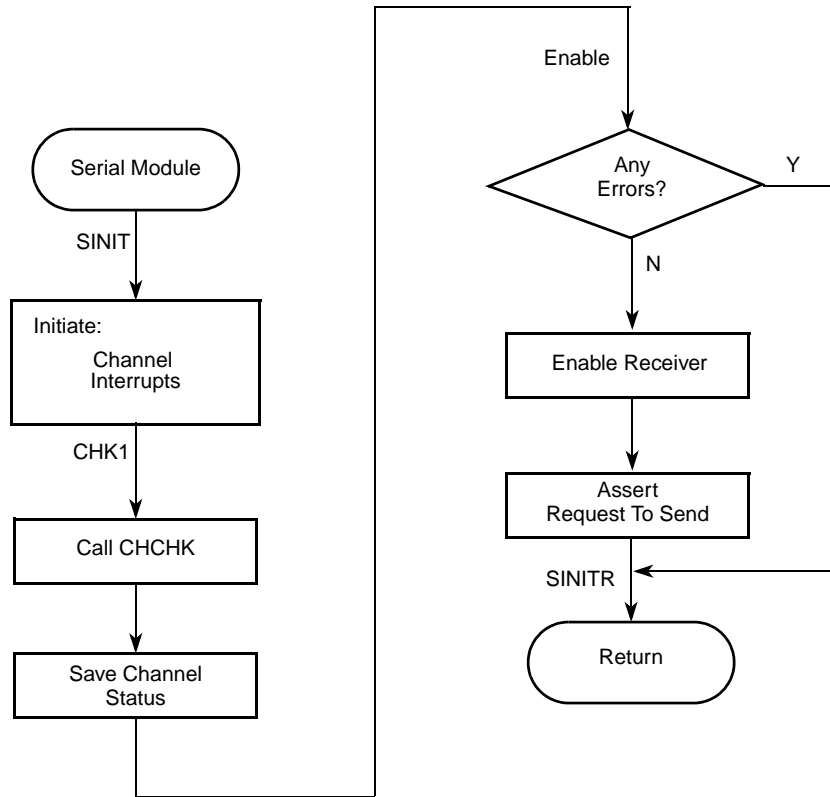
Figure 31-25. UART Mode Programming Flowchart (Sheet 1 of 5)
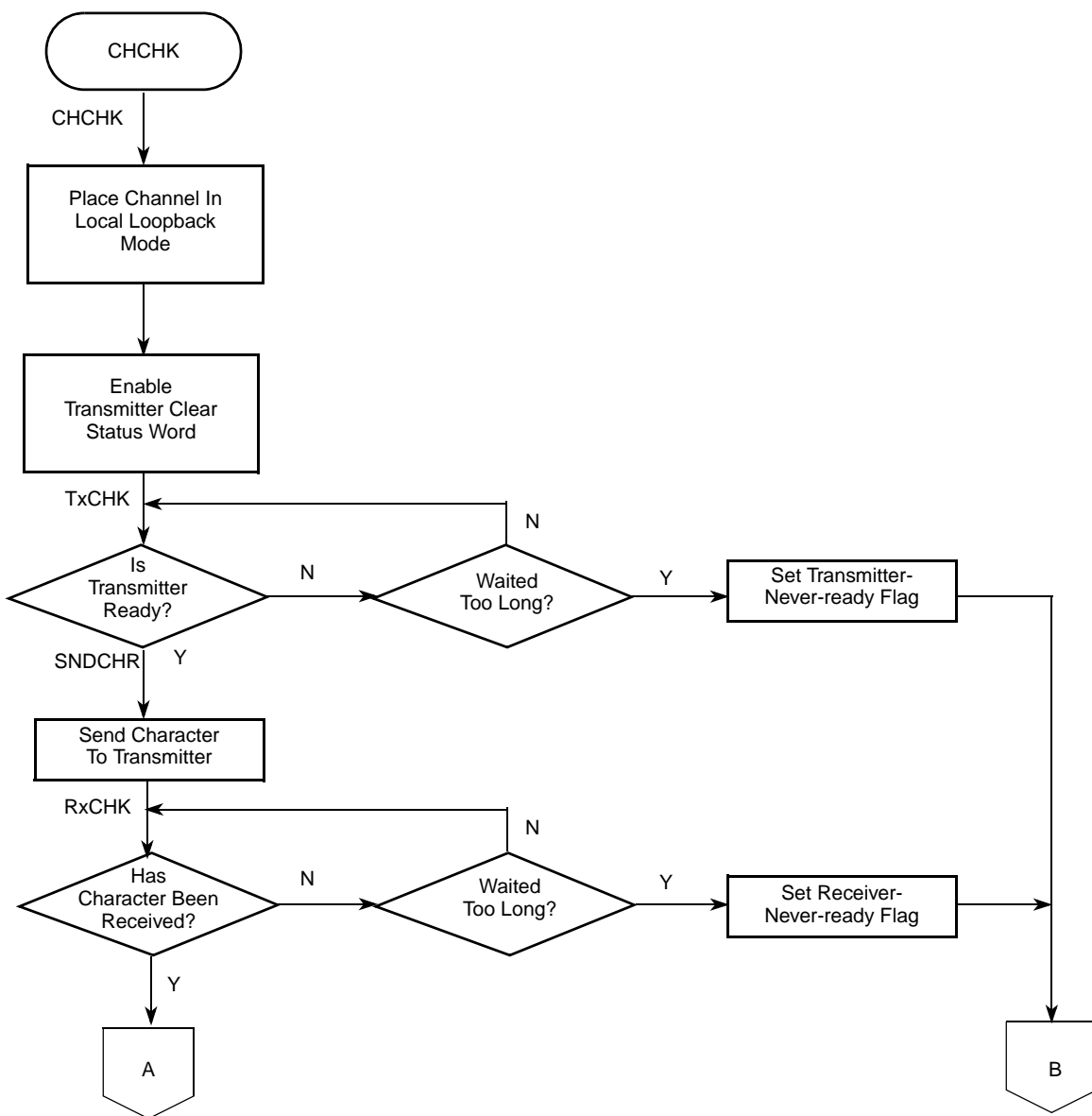
Figure 31-25. UART Mode Programming Flowchart (Sheet 2 of 5)
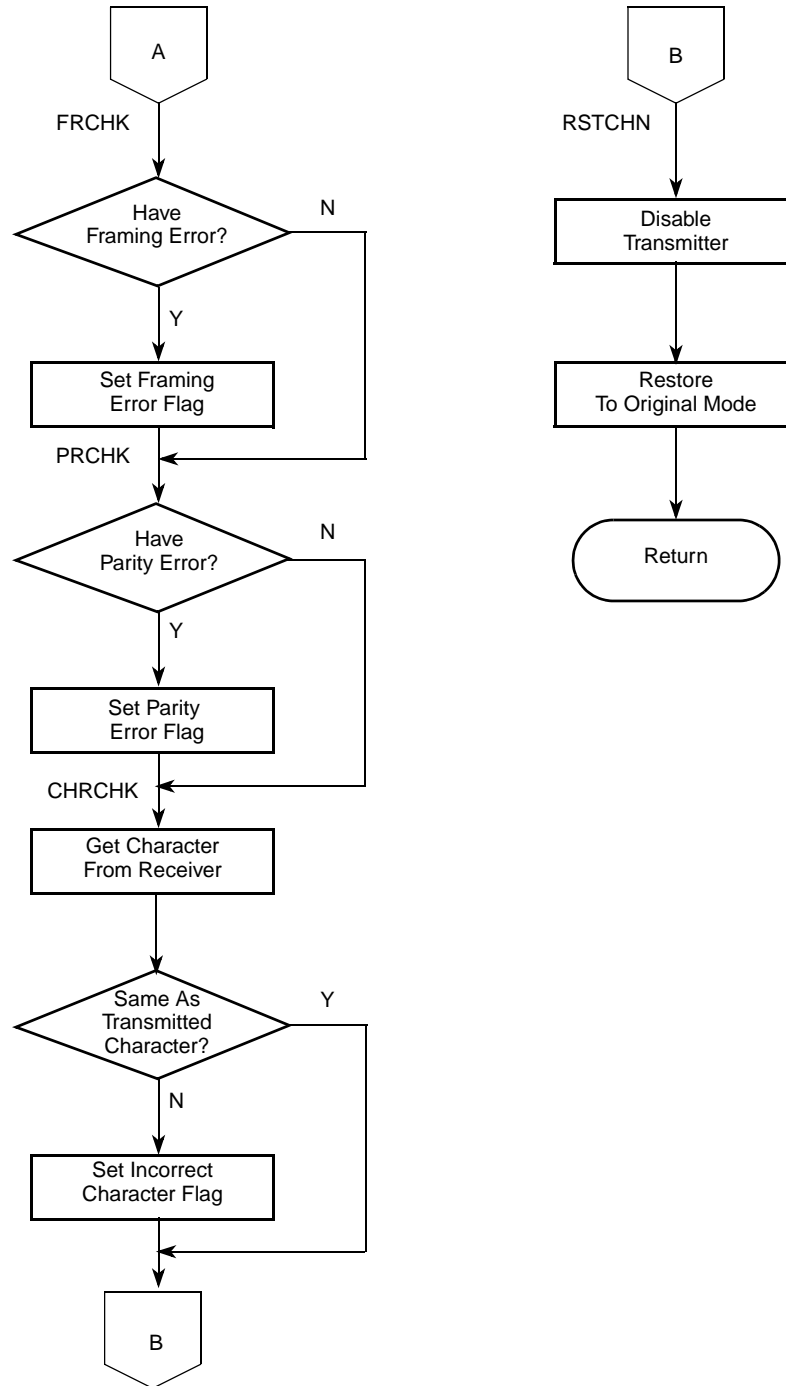
Figure 31-25. UART Mode Programming Flowchart (Sheet 3 of 5)
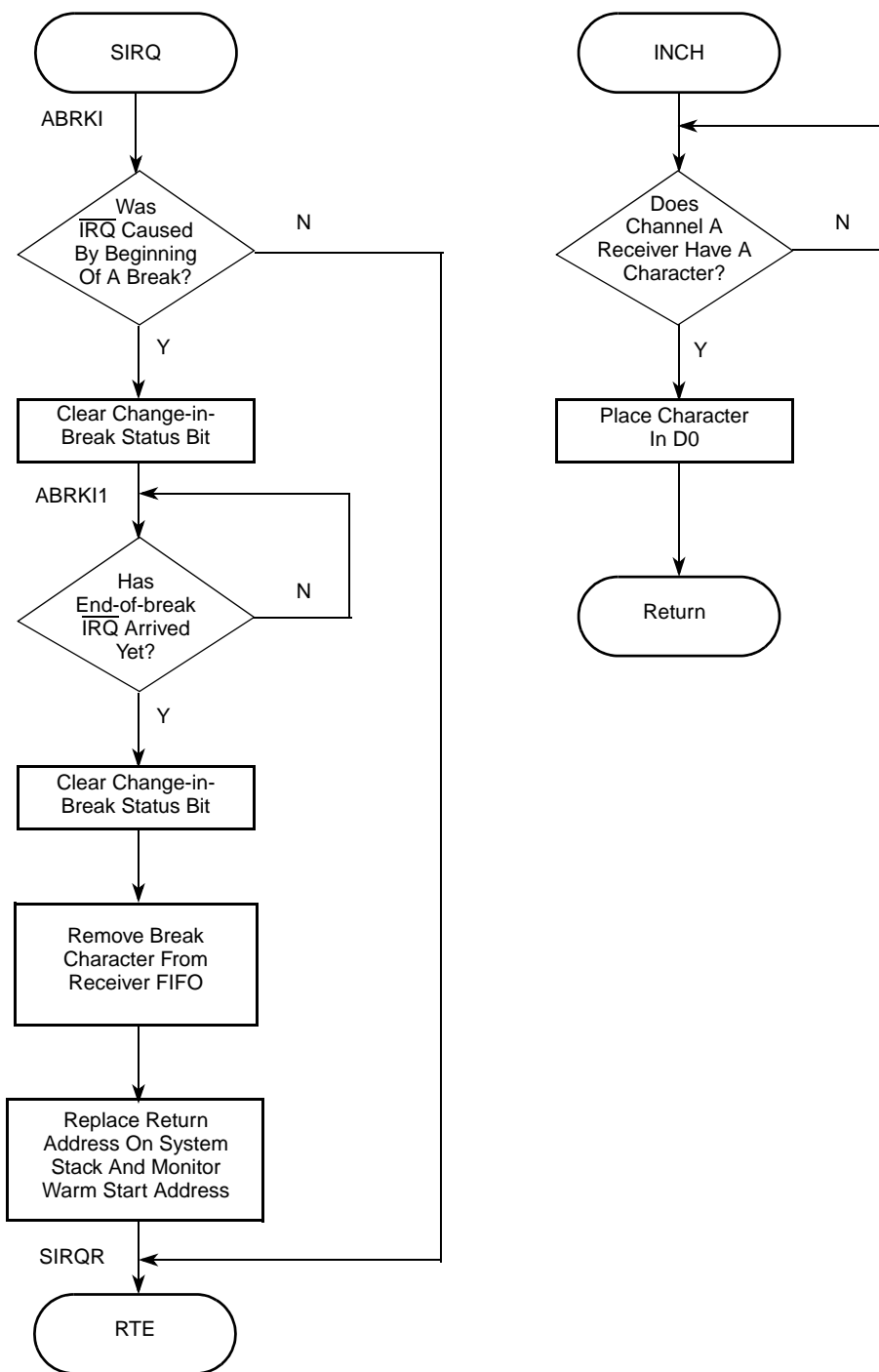
Figure 31-25. UART Mode Programming Flowchart (Sheet 4 of 5)
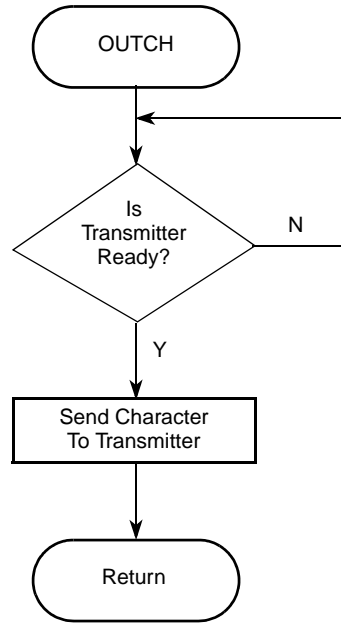
Figure 31-25. UART Mode Programming Flowchart (Sheet 5 of 5)

Chapter 32
I^{2}C Interface

32.1 Introduction

This chapter describes the I^{2}C module, clock synchronization, and I^{2}C programming model registers. It also provides extensive programming examples.

32.1.1 Block Diagram

Figure 32-1 is a I^{2}C module block diagram, illustrating the interaction of the registers described in Section 32.2, "Memory Map/Register Definition".
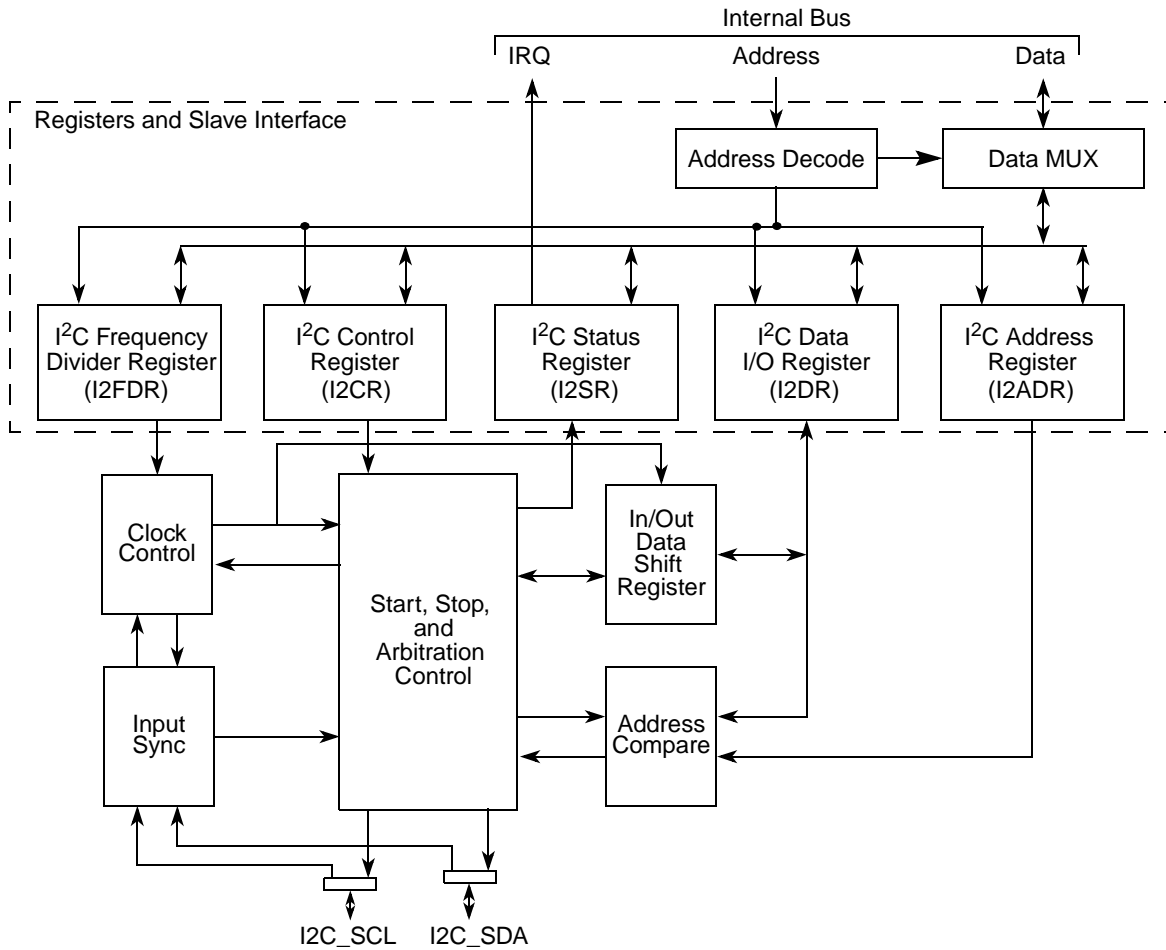
Figure 32-1. I^{2}C Module Block Diagram

32.1.2 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 feature 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 13, "General Purpose I/O Module") prior to configuring the I^{2}C module.

32.1.3 Features

The I^{2}C module has these key features:

- Compatibility with I^{2}C bus standard version 2.1
- Support for 3.3-V tolerant devices
- 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

32.2 Memory Map/Register Definition

The below table lists the configuration registers used in the I²C interface.

Table 32-1. I²C Module Memory Map

Address	Register	Access	Reset Value	Section/Page
0xFC05_8000	I²C Address Register (I2ADR)	R/W	0x00	32.2.1/32-3
0xFC05_8004	I²C Frequency Divider Register (I2FDR)	R/W	0x00	32.2.2/32-3
0xFC05_8008	I²C Control Register (I2CR)	R/W	0x00	32.2.3/32-4
0xFC05_800C	I²C Status Register (I2SR)	R/W	0x81	32.2.4/32-5
0xFC05_8010	I²C Data I/O Register (I2DR)	R/W	0x00	32.2.5/32-6

32.2.1 I²C Address Register (I2ADR)

I2ADR holds the address the I²C 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
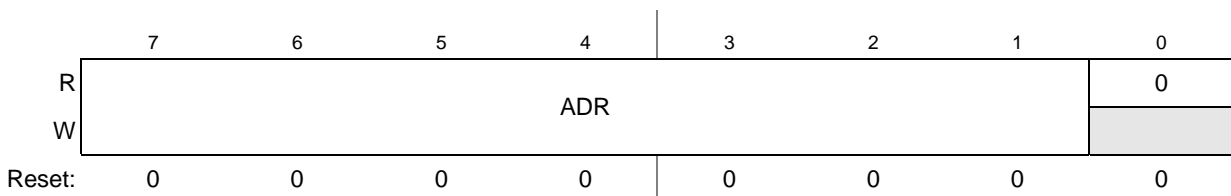
Figure 32-2. I²C Address Register (I2ADR)

Table 32-2. I2ADR Field Descriptions

Field	Description
7–1 ADR	Slave address. Contains the specific slave address to be used by the I²C module. Slave mode is the default I²C mode for an address match on the bus.
0	Reserved, must be cleared.

32.2.2 I²C Frequency Divider Register (I2FDR)

The I2FDR, shown in Figure 32-3, provides a programmable prescaler to configure the I²C 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
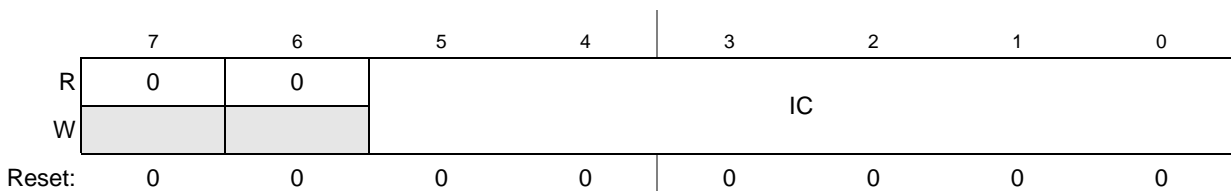
Figure 32-3. I²C Frequency Divider Register (I2FDR)

Table 32-3. I2FDR Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 IC	I²C 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

32.2.3 I²C Control Register (I2CR)

I2CR enables the I²C module and the I²C 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
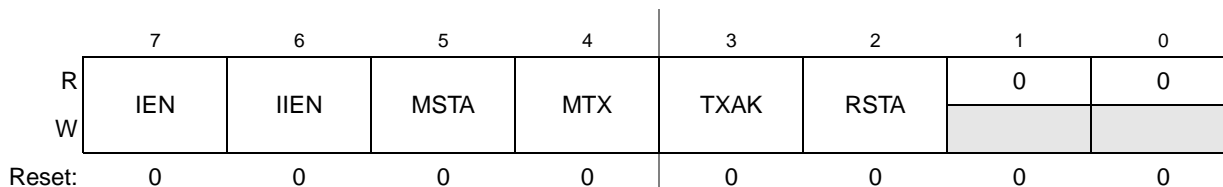
Figure 32-4. I²C Control Register (I2CR)

Table 32-4. I2CR Field Descriptions

Field	Description
7 IEN	I²C enable. Controls the software reset of the entire I²C 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 I²C module to lose arbitration, after which bus operation returns to normal. 0 The I²C module is disabled, but registers can be accessed. 1 The I²C module is enabled. This bit must be set before any other I2CR bits have any effect.
6 IIEN	I²C interrupt enable. 0 I²C module interrupts are disabled, but currently pending interrupt condition is not cleared. 1 I²C module interrupts are enabled. An I²C 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 I²C 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	Reserved, must be cleared.

32.2.4 I²C 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
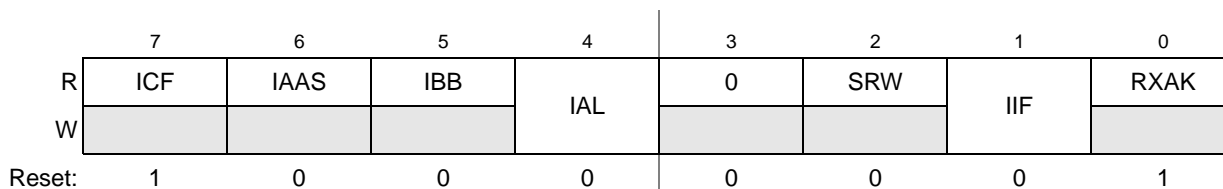
Figure 32-5. I²C Status Register (I2SR)

Table 32-5. I2SR Field Descriptions

Field	Description
7 ICF	I^{2}C 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	I^{2}C 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	I^{2}C 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	I^{2}C 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 I^{2}C 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	I^{2}C interrupt. Must be cleared by software by writing a 0 in the interrupt routine. 0 No I^{2}C 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.

32.2.5 I^{2}C 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 I^{2}C 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
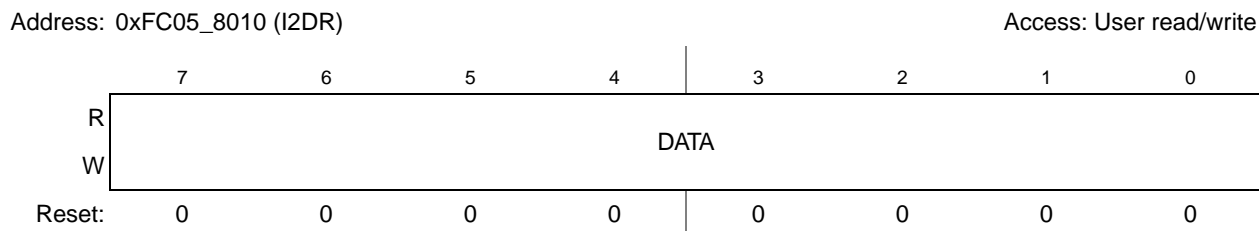
Figure 32-6. I²C Data I/O Register (I2DR)

Table 32-6. I2DR Field Description

Field	Description
7–0 DATA	I²C 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.

32.3 Functional Description

The I²C module uses a serial data line (I2C_SDA) and a serial clock line (I2C_SCL) for data transfer. For I²C 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 I²C default state is as a slave receiver. Therefore, when not programmed to be a master or responding to a slave transmit address, the I²C module should return to the default slave receiver state. See Section 32.4.1, "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.

32.3.1 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 32-7). 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.
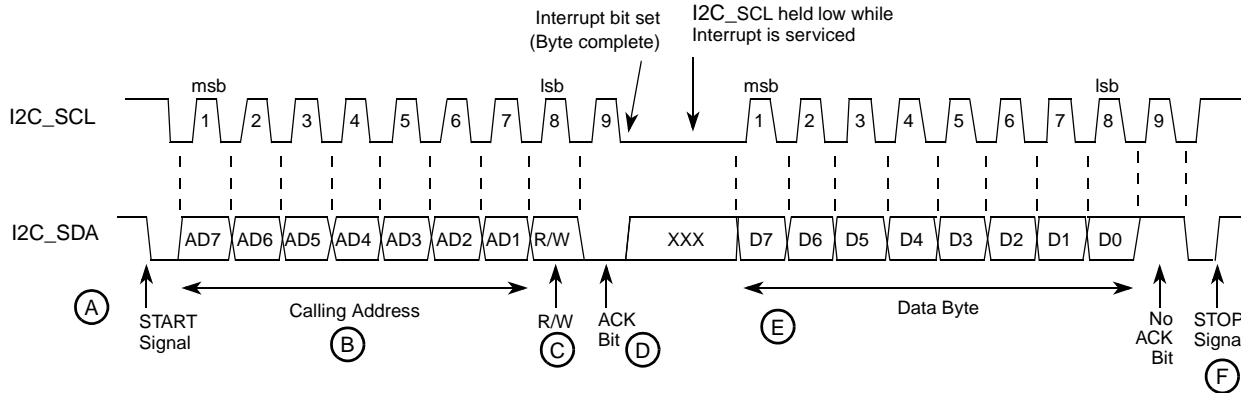
Figure 32-7. I²C Standard Communication Protocol

32.3.2 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.

32.3.3 Data Transfer

When successful slave addressing is achieved, data transfer can proceed (see E in Figure 32-7) 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 32-7 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 32-8.
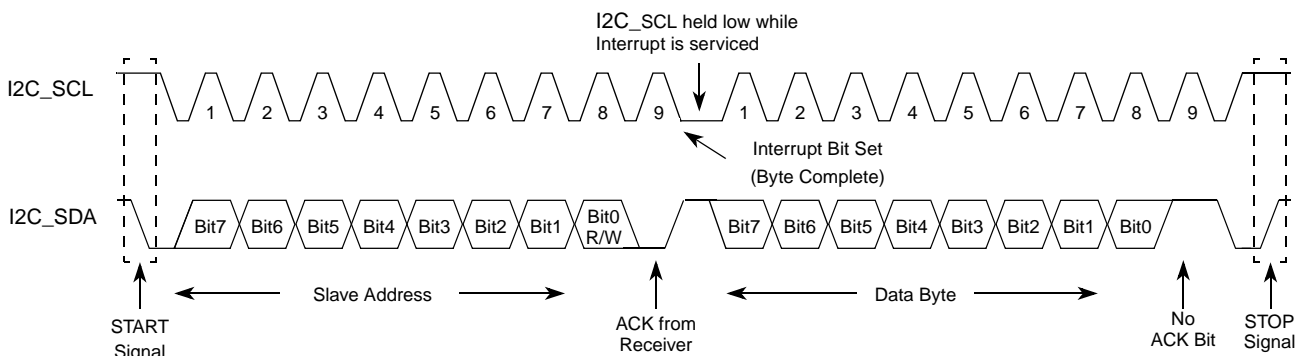
Figure 32-8. Data Transfer

32.3.4 Acknowledge

The transmitter releases the I2C_SDA line high during the acknowledge clock pulse as shown in Figure 32-9. 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 32-10 and discussed in Section 32.3.6, "Repeated START") to start a new calling sequence.
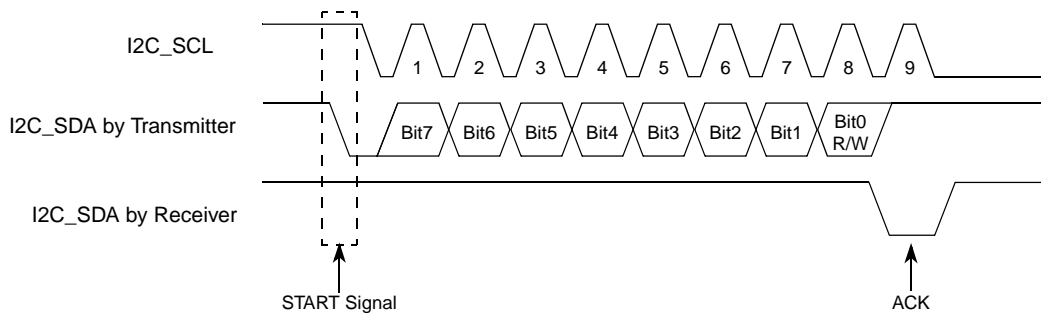
Figure 32-9. Acknowledgement by Receiver

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 32-9).

32.3.5 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 32-7). 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 32.3.6, "Repeated START."

32.3.6 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 32-10. 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.
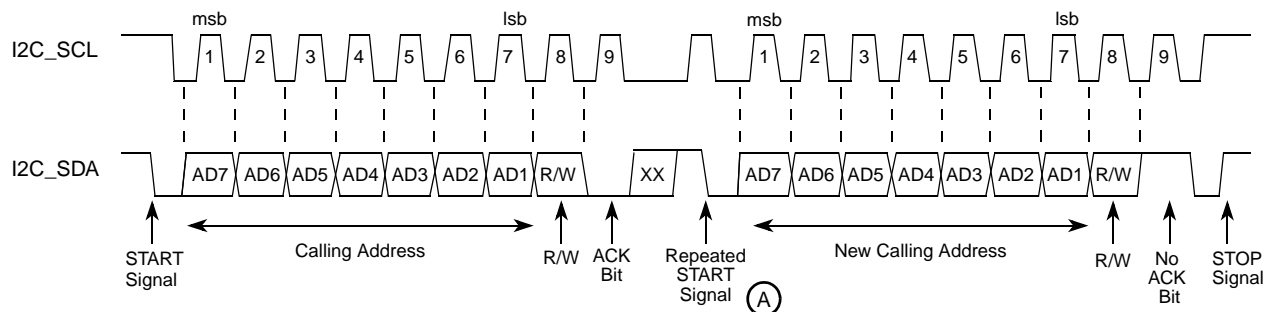
Figure 32-10. Repeated START

Various combinations of read/write formats are then possible:

- The first example in Figure 32-11 is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.

- The second example in Figure 32-11 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 32-11, 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/$\overline{\text{W}}$ bit.
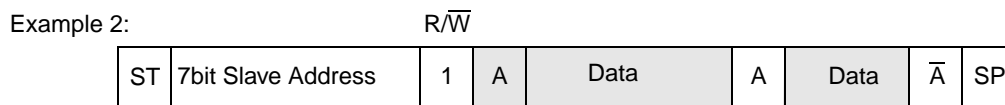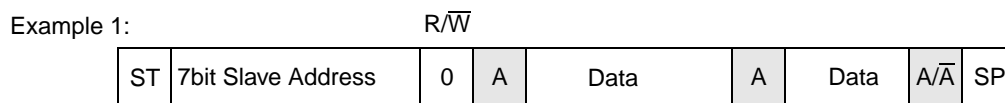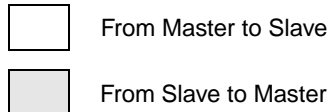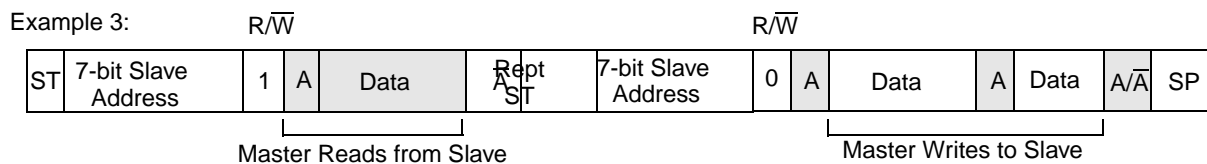
Figure 32-11. Data Transfer, Combined Format

32.3.7 Clock Synchronization and Arbitration

I²C 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 32-12). 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.
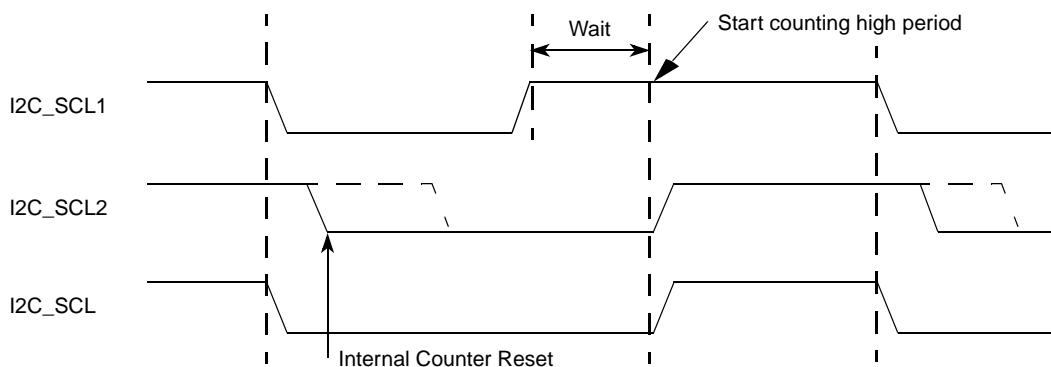
Figure 32-12. Clock Synchronization

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 32-13). 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.
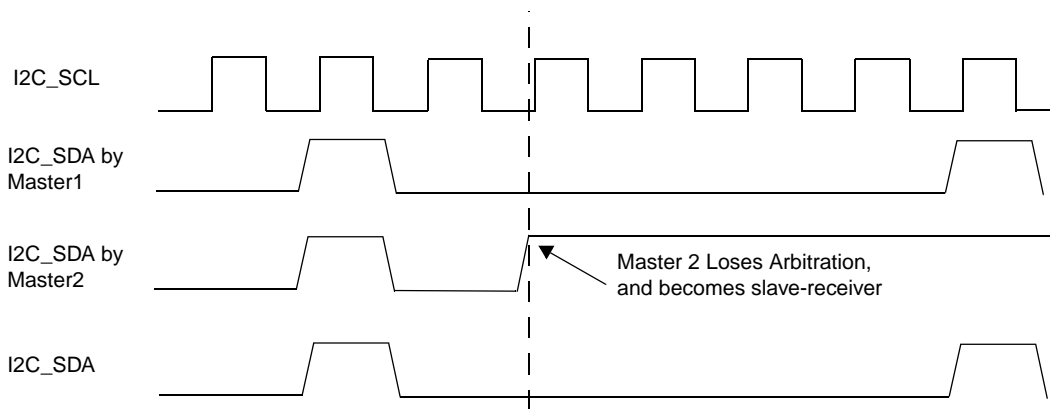
Figure 32-13. Arbitration Procedure

MCF5329 Reference Manual, Rev 3

32.3.8   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.

32.4   Initialization/Application Information

The following examples show programming for initialization, signaling START, post-transfer software response, signaling STOP, and generating a repeated START.

32.4.1   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 32.2.2, "I2C Frequency Divider Register (I2FDR)."
2. Update the I2ADR to define its slave address.
3. Set I2CR[IEN] to enable the I²C 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 I²C 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
```

32.4.2   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.

32.4.3   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 I²C 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 32-14).

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.

32.4.4   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.

32.4.5 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.

32.4.6 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.

32.4.7 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.
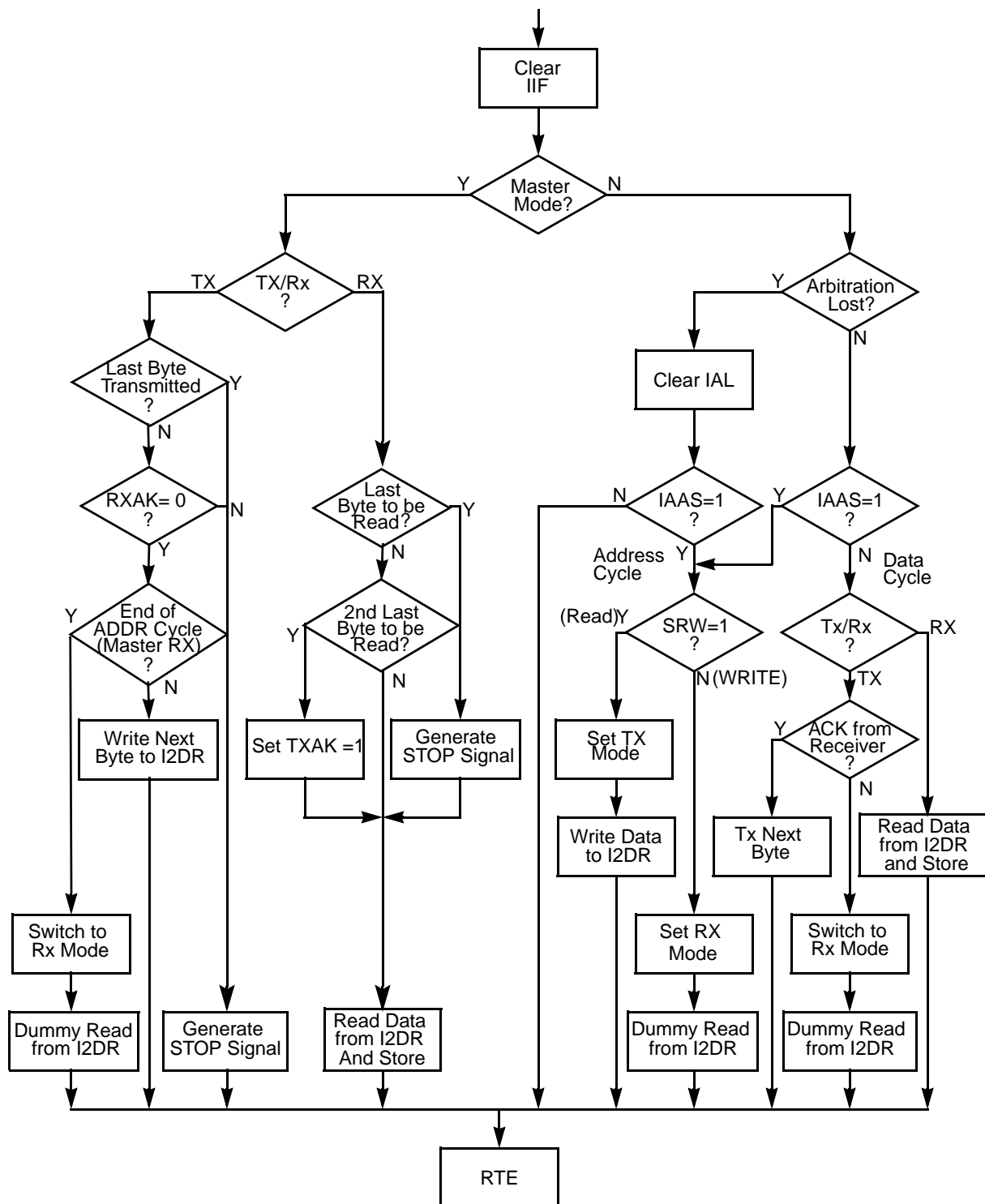
Figure 32-14. Flow-Chart of Typical I²C Interrupt Routine

Chapter 33
Message Digest Hardware Accelerator (MDHA)

33.1 Introduction

This chapter describes the message digest hardware accelerator (MDHA).

NOTE

The MCF5327, MCF5328, and MCF53281 do not contain cryptography modules. Refer to Table 1-1 for details on device configurations.

33.1.1 Overview

The MDHA is a hardware implementation of two of the world's most popular cryptographic hash functions: SHA-1 (Secure Hash Algorithm 1) and MD5 (Message Digest 5). SHA-1 and MD5 are critical algorithms to the IPSec (IP Security) Protocol, a fast-spreading protocol for authentication and encryption over the internet in networking equipment. The MDHA also includes circuitry to automate the process of generating an HMAC (hashed message authentication code) as specified by RFC 2104 and EHMAC (enhanced hashed message authentication code), using only the SHA-1 algorithm.

33.1.2 Features

The MDHA computes a single message digest (or hash or integrity check) value of all the data presented on the input bus, using the MD5 or SHA-1 algorithms for bulk data hashing. The MDHA includes these distinctive features:

- MD5 one-way 128-bit hash function specified in RFC 1321.
- SHA-1 one-way 160-bit hash function specified by the ANSI X9.30-2 and FIPS 180-1 standards.
- HMAC support for all algorithms, as specified in RFC 2104.
- EHMAC support for the SHA-1 algorithm.
- EHMAC key support up to 160 bits.
- Processes 512-bit blocks organized as 16×32 bit longwords.
- Automatic message and key padding.
- Internal 16x32 bit FIFO for temporary storage of hashing data.

With any hash algorithm, the larger message is mapped onto a smaller output space. Therefore collisions are potential, albeit not probable. The 160-bit hash value is a sufficiently large space such that collisions are extremely rare. The security of the hash function is based on the difficulty of locating collisions. That is, it is computationally infeasible to construct two distinct but similar messages that produce the same hash output.

This module is useful in many applications including hashing messages to generate digital signatures or computation of a shared secret. The digital signature is typically computed on a small input, however if the data to be signed is large, it is inefficient to sign the entire data. Instead, the large input data is hashed to a smaller value which is then signed. If the message is also sent to the verifying authority along with the signature, the verifying authority can verify the signature by recovering the hash value from the signature using the public key of the sender, hashing the message itself, and then comparing the computed hash value with the recovered hash value. If they match, then the verifying authority is confident that the data was signed by the owner of the private key that matches the public key, where the private key presumably is only known by the sender. This provides a measure of authentication and non-repudiation.

A conceptual block diagram of the MDHA module is shown in Figure 33-1. Multiple input blocks are written to the MDHA module, and at the end, the hash value is read as the 160-bit output for SHA-1 or 128-bit output for MD5.
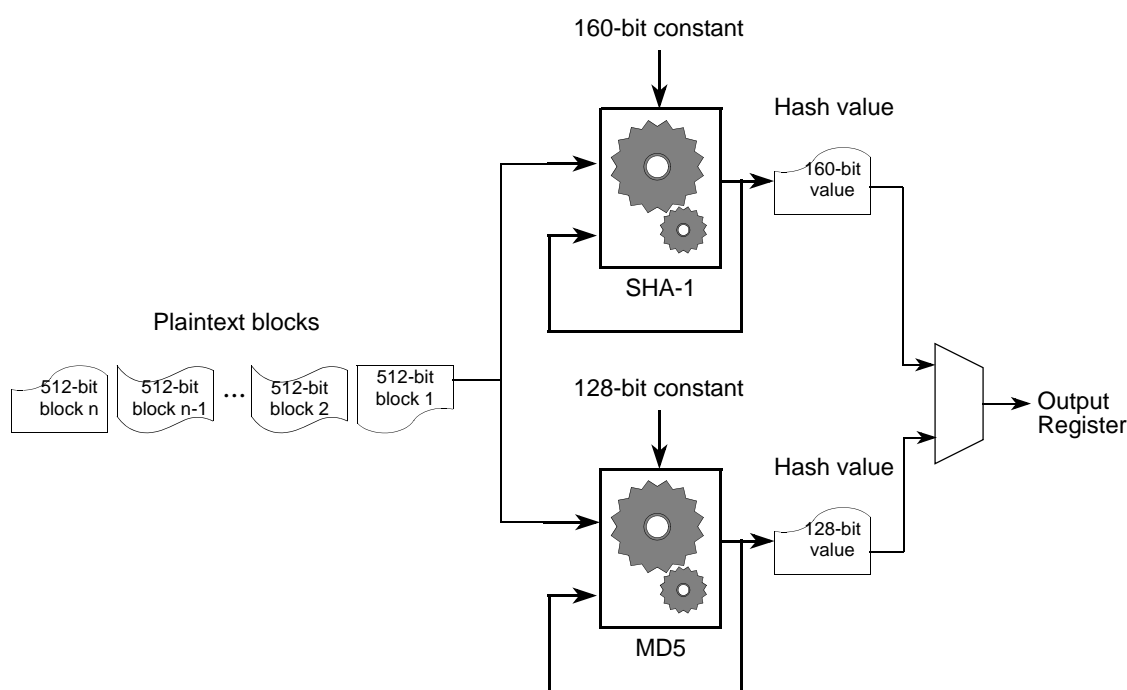
Figure 33-1. MDHA Hashing Process

33.1.3 Modes of Operation

- One-way hashing generation—a message digest, or a hash, is the result of a one-way function performed on a message. Two different hash functions can be used:
  - SHA-1: Secure Hash Algorithm defined in Federal Information Processing Standards Publication (FIPS) 180-1.
  - MD5: Message Digest 5 algorithm defined by Ron Rivest of MIT Laboratory for Computer Science and RSA Data Security, INC.
- MAC: A Message authentication code is a one-way function performed on a message with two user defined keys:

— HMAC—Hashed message authentication can be used with the SHA-1 or MD5 algorithm defined in FIPS 198.

— EHMAC—Enhanced Hashed message authentication can only be used with the SHA-1 algorithm.

33.2   Memory Map/Register Definition

Table 33-1. MDHA Module Memory Map

Address	Register	Width (bits)	Access[1]	Reset Value	Section/Page
0xEC08_0000	MDHA Mode Register (MDMR)	32	R/W	0x0000_0000	33.2.1/33-3
0xEC08_0004	MDHA Control Register (MDCR)	32	R/W	0x0000_0000	33.2.2/33-6
0xEC08_0008	MDHA Command Register (MDCMR)	32	W	0x0000_0000	33.2.3/33-7
0xEC08_000C	MDHA Status Register (MDSR)	32	R	0x0000_8408	33.2.4/33-8
0xEC08_0010	MDHA Interrupt Status Registers (MDISR)	32	R	0x0000_0000	33.2.5/33-9
0xEC08_0014	MDHA Interrupt Mask Registers (MDIMR)	32	R/W	0x0000_0000	33.2.5/33-9
0xEC08_001C	MDHA Data Size Register (MDDSR)	32	R/W	0x0000_0000	33.2.6/33-11
0xEC08_0020	MDHA Input FIFO (MDIN)	32	W	0x0000_0000	33.2.7/33-11
0xEC08_0030	MDHA Message Digest A0 Register (MDA0)	32	R/W	0x0123_4567	33.2.8/33-11
0xEC08_0034	MDHA Message Digest B0 Register (MDB0)	32	R/W	0x89AB_CDEF	33.2.8/33-11
0xEC08_0038	MDHA Message Digest C0 Register (MDC0)	32	R/W	0xFEDC_BA98	33.2.8/33-11
0xEC08_003C	MDHA Message Digest D0 Register (MDD0)	32	R/W	0x7654_3210	33.2.8/33-11
0xEC08_0040	MDHA Message Digest E0 Register (MDE0)	32	R/W	0xF0E1_D2C3	33.2.8/33-11
0xEC08_0044	MDHA Message Data Size Register (MDMDS)	32	R/W	0x0000_0000	33.2.9/33-12
0xEC08_0070	MDHA Message Digest A1 Register (MDA1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_0074	MDHA Message Digest B1 Register (MDB1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_0078	MDHA Message Digest C1 Register (MDC1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_007C	MDHA Message Digest D1 Register (MDD1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_0080	MDHA Message Digest E1 Register (MDE1)	32	R/W	0x0000_0000	33.2.10/33-12

[1]   Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

33.2.1   MDHA Mode Register (MDMR)

The MDMR stores the current processing mode. It can be written before a hashing operation begins. After the hashing operation has begun, an error is generated if the register is written. This register is reset only via a hardware or software reset.

Address: 0xEC08_0000 (MDMR)   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	SSL	MAC FULL	SWAP	OPAD	IPAD	INIT	MAC		PDATA	0	ALG
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-2. MDHA Mode Register (MDMR)

Table 33-2. MDMR Field Descriptions

Field	Description
31–11	Reserved, should be cleared.
10 SSL	Secure socket layer MAC. Implements the SSL 2.0/3.0 defined MAC. Only applicable for the MD5 algorithm with the MACFULL bit set. 0 Do not perform SSL. 1 Perform SSL
8 SWAP	Message authentication code full. Allows the user to input a key and message data and have the accelerator do the complete MAC in one step. Used directly with HMAC or EHMAC mode. 0 Do not perform MAC FULL. 1 Perform MAC FULL.
7 OPAD	Swap message digest. For SHA-1 only. Swap the output direction of the Message Digest data. The data registers are reversed and byte swapped. This allows for viewing data in the reverse order which might be used by other algorithms. See the table below for an example. 0 Do not perform swap. 1 Swap output direction. _(see table below)_
6 IPAD	Inner padding of message. Exclusive OR the message with 0x3636_3636. Hash used with HMAC. Requires key to be loaded into the FIFO 0 Do not perform padding 1 Perform padding

Table within OPAD field description:

Algorithm	Message Digest Register	SWAP = 0				SWAP = 1			
SHA-1	A0	D	C	B	A	Q	R	S	T
	B0	H	G	F	E	M	N	O	P
	C0	L	K	J	I	I	J	K	L
	D0	P	O	N	M	E	F	G	H
	E0	T	S	R	Q	A	B	C	D

Table 33-2. MDMR Field Descriptions (continued)

Field	Description
5 INIT	Initialization. Performs algorithm specific initialization of the digest registers. Most operations require this bit to be set. Only static operations that are continuing from a known intermediate hash value should clear this bit. 0 Do not perform initialization 1 Initialize the selected algorithm's starting registers
4–3 MAC	Message authentication code. Performs message authentication on messages. Requires keys loaded into the context and key registers. 00 Do not perform MAC 01 Perform HMAC 10 Perform EHMAC 11 Reserved
2 PDATA	Pad data bit. Performs automatic message padding on the current partial message block. 0 Do not perform padding 1 Perform padding
1	Reserved, should be cleared.
0 ALG	Algorithm. Selects which algorithm the MDHA module uses 0 Secure Hash Algorithm (SHA-1) 1 Message Digest 5 (MD5)

33.2.1.1   Invalid Modes

The following mode combinations trigger an interrupt to the interrupt controller and set the mode error bit in the MDHA interrupt status register.

Table 33-3. Invalid MDMR Bit Settings

MDMR bit settings		Comments
Setting any reserved bits.		—
IPAD=1	OPAD=1	Asserting both of the signals at the same time causes a mode to occur that the MDHA is not capable of performing. These two modes must be performed separately.
IPAD=1	PDATA=1	According to HMAC and EHMAC standards no padding is done to the data when the IPAD function is performed.
OPAD=1	PDATA=1	According to HMAC and EHMAC standards no padding is done to the data when the OPAD function is performed.
MAC=10 (EHMAC)	IPAD=1	MDHA requires that the IPAD step be performed as a separate hash operation than message authentication.
MAC=10 (EHMAC)	OPAD=1	MDHA requires that the OPAD step be performed as a separate hash operation than message authentication.
MAC=10 (EHMAC)	ALG=1 (MD5)	The standard for EHMAC is only defined for the SHA-1 algorithm.
MAC=01 (HMAC)	IPAD	MDHA requires that the IPAD step be performed as a separate hash operation than message authentication.
MAC=01 (HMAC)	OPAD	MDHA requires that the OPAD step be performed as a separate hash operation than message authentication.

Table 33-3. Invalid MDMR Bit Settings

MDMR bit settings		Comments
SSL=1	ALG=0 (SHA-1)	SSL MAC is only functional for the MD5 algorithm.
SWAP=1	ALG=1 (MD5)	The SWAP bit is designed to support a particular function for the SHA-1 algorithm and is invalid for MD5.

33.2.2 MDHA Control Register (MDCR)

The MDCR contains bits that should be set following a hardware reset.

Address: 0xEC08_0004 (MDCR)   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			DMAL			0	0	0	0	0	0	0	0	0	0	0	0	0	END	DMA	IE
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
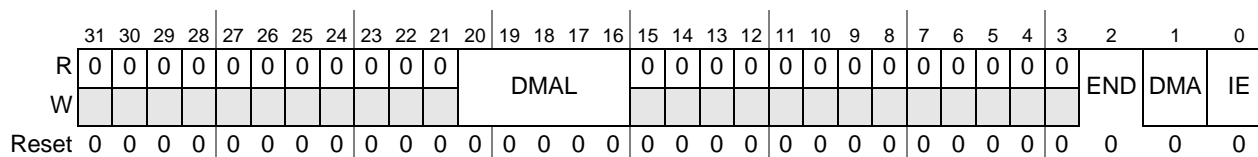
Figure 33-3. MDHA Control Register (MDCR)

Table 33-4. MDCR Field Descriptions

Field	Description
31–21	Reserved, should be cleared.
20–16 DMAL	DMA request level. Represents the minimum number of words available in the FIFO between DMA requests. For example: When DMAL is programmed to 5 longwords, the DMA request signal is not asserted until there is space in the input FIFO for 5 longwords. This value must be 16 longwords or less. 00001–10000 = 1–16 words All others reserved.
15–3	Reserved, should be cleared.
2 END	Endian select 0 Little endian mode 1 Big endian mode
1 DMA	DMA enable. Enables/disables DMA requests from the MDHA module. 0 Disable DMA requests 1 Enable DMA requests
0 IE	Interrupt enable. Enables/disables interrupts from the MDHA module. 0 Disable interrupts 1 Enable interrupts

33.2.3   MDHA Command Register (MDCMR)

The MDCMR register is used for software control of hashing and resetting. This register always reads zeros.

Address: 0xEC08_0008 (MDCMR)   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	0	0	0	0	0	0	0	0	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																													GO	CI	RI	SWR
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 33-4. MDHA Command Register (MDCMR)

Table 33-5. MDCMR Field Descriptions

Field	Description
31–4	Reserved, should be cleared.
3 GO	Go. Indicates that all data has been loaded into the input FIFO and the module should complete all processing. This bit is self clearing. 0 Do not complete all processing. 1 Finish all processing.
2 CI	Clear IRQ. Clears errors in the MDISR register and deasserts any interrupt requests from the MDHA module. This bit is self clearing. 0 Do not clear interrupts & errors. 1 Clear interrupts & errors.
1 RI	Re-initialize. Re-initializes memory and clears all registers except the MDESM and MDCR. 0 No re-initialization 1 Re-initialize the MDHA module
0 SWR	Software reset. Resets all registers and re-initialize memory of the MDHA. Functionally equivalent to hardware reset. This bit is self clearing. 0 No reset 1 Software reset

33.2.4 MDHA Status Register (MDSR)

The MDSR stores the current status of the MDHA. This register is used to debug errors and to give a view into the workings of the MDHA's internal engines.

Address: 0xEC08_000C (MDSR)   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	IFL							
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	APD			0	0	FS			GNW	HSH	0	BUSY	RD	ERR	DONE	INT
W																
Reset	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
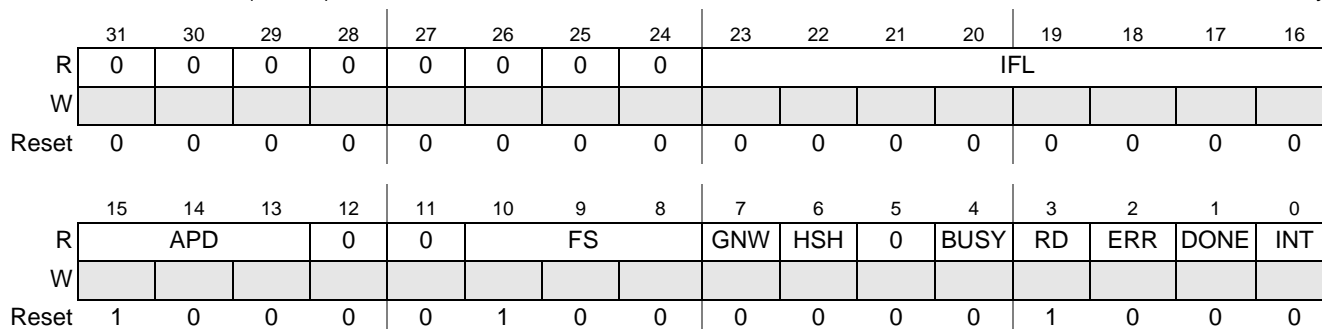
Figure 33-5. MDHA Status Register (MDSR)

Table 33-6. MDSR Field Descriptions

Field	Description
31–24	Reserved, should be cleared.
23–16 IFL	Input FIFO level. Read-only. The current number of longwords that are in the Input FIFO. IFL ranges from 0–16 longwords (0x00-0x10).
15–13 APD	Auto pad state. Read-only. Indicates the current state of the autopadder for debug purposes. 000 Perform standard auto padding. 001 Pad last word. 010 Add a word for padding. 011 Last hash for the EHMAC. 100 Stall state (Auto Padder passes no data to engine). This is the default state that the module enters whenever there is an error. All other settings are reserved.
12–11	Reserved, should be cleared.
10–8 FS	FIFO size. Read-only. Indicates the size of the internal FIFO, which is fixed at 16 longwords for this device. 100 16 longwords
7 GNW	Get next word. Read-only. Indicates that the MDHA engine has not filled an entire block and is requesting more data. 0 Does not need any data 1 Requesting more data
6 HSH	Hashing. Read-only. Indicates that data is currently being hashed 0 Waiting for more data 1 Hashing current data
5	Reserved, should be cleared.
4 BUSY	Busy. Read-only. Indicates that the module is busy processing data. 0 Idle or done 1 Busy processing data
3 RD	Reset interrupt. Read-only. Indicates the MDHA module has completed resetting. 0 Reset in progress 1 Completed reset sequence

Table 33-6. MDSR Field Descriptions (continued)

Field	Description
2 ERR	Error interrupt. Read-only. Indicates that an error has occurred. Set if any bit in the MDISR is set. 0 No Error 1 Error has occurred
1 DONE	Done interrupt. Read-only. Indicates that the MDHA module has completed processing the requested amount of data. 0 Not complete 1 Done processing
0 INT	MDHA single interrupt. Read-only. Indicates that the MDHA module has finished processing the message and the hash result is ready to be read from the message digest register or there is an error. 0 No interrupt 1 Done or error interrupt

33.2.5 MDHA Interrupt Status & Mask Registers (MDISR and MDIMR)

The MDISR describes the current error that has taken place. An interrupt request is generated when any one of these bits is set unless the corresponding bit is set in the MDIMR. MDISR is only cleared after a hardware or software reset has been performed. The bit definitions for MDISR and MDIMR are similar.

NOTE
Masking errors should only be used for debug purposes. A masked error most likely causes invalid data.

Address: 0xEC08_0010 (MDISR)   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	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	DRL	GTDS	ERE	RMDP	0	DSE	IME	0	NEIF	0	IFO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-6. MDHA Interrupt Status Registers (MDISR)

Address: 0xEC08_0014 (MDIMR)   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	DRL	GTDS	ERE	RMDP	0	DSE	IME	0	NEIF	0	IFO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-7. MDHA Interrupt Mask Registers (MDIMR)

Table 33-7. MDISR & MDIMR Field Descriptions

Field	Description
31–11	Reserved, should be cleared.
10 DRL	DMA request level error. Indicates that the MDCR[DMAL] field was set to an invalid size and DMA requests are enabled (MDCR[DMA] = 1). The request level must be set to a value of 1-16 words. Zero is considered to be an invalid request level 0 No error 1 Invalid DMA request level
9 GTDS	Greater than data size error. Read only. Indicates that the GO bit was set in the MDCR and the data size written to the MDDSR is greater then the amount of data written to the FIFO. 0 No error 1 Datasize is greater than the message size
8 ERE	Early read error. Read only. A context register was read from while the module was busy processing data. 0 No error 1 Early read error
7 RMDP	Register modified during processing. Read only. An MDHA register was modified while the module was busy processing data. 0 No error 1 Register modified
6	Reserved, should be cleared.
5 DSE	Illegal data size. Read only. Illegal data size was written to the MDHA Data Size Register (MDDSR). Data size written into the MDDSR is greater than the allocated size. 0 No error 1 Illegal data size in MDDSR
4 IME	Illegal mode interrupt. Read only. Illegal mode is set in the MDMR. Consult Section 33.2.1.1, "Invalid Modes," for more information on invalid modes. 0 No error 1 Illegal value in MDMR
3	Reserved, should be cleared.
2 NEIF	Non-empty input FIFO upon done. Read only. The Input FIFO contained data when processing was completed 0 No error 1 FIFO contained data when finished processing

Field	Description
1	Reserved, should be cleared.
0 IFO	Input FIFO Overflow. Read only. The Input FIFO has been written to while full. 0 No overflow occurred 1 Input FIFO overflow error

33.2.6 MDHA Data Size Register (MDDSR)

The MDDSR stores the size of the last block of data to be processed. This value is in bytes. The first two bits are used to identify the ending byte location in the last word. This is used to add the data padding when auto padding is selected in the MDMR. Load this register with the amount of data to be processed in the FIFO. This register is cleared when the MDHA is reset, re-initialized and at the end of processing the complete message.

Address: 0xEC08_001C (MDDSR)   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														MDHA Data Size															
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
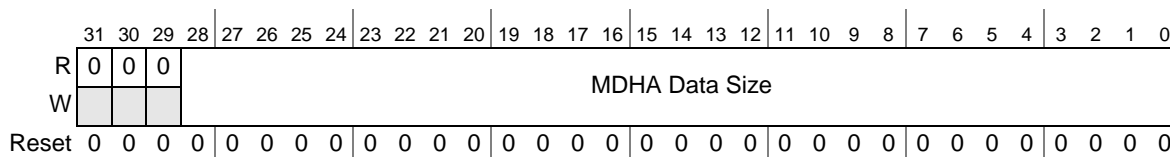
Figure 33-8. MDHA Data Size Register (MDDSR)

33.2.7 MDHA Input FIFO (MDIN)

The MDIN provides temporary storage for data to be used during hashing. The FIFO is a write only register and attempting to read from this register always returns 0. If the FIFO is written to when the FIFO Level is full, an interrupt request is generated and the MDISR[IFO] bit is set. The MDSR[IFL], described in Section 33.2.4, "MDHA Status Register (MDSR)," can be polled to monitor how many 32-bit longwords are currently resident in the FIFO.

Address: 0xEC08_0020 (MDIN)   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	0	0	0	0	0	0	0	0	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													Input FIFO																			
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
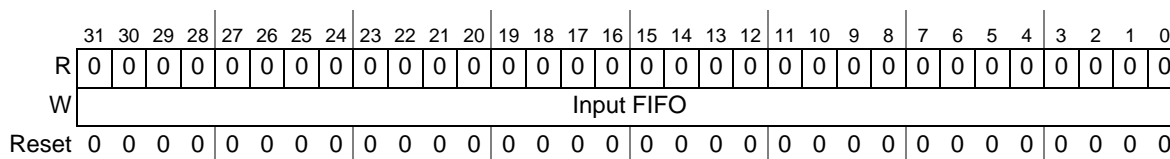
Figure 33-9. MDHA Input FIFO (MDIN)

33.2.8 MDHA Message Digest Registers 0 (MD*x*0)

The MDHA message digest registers 0 consist of five 32-bit registers (MDA0, MDB0, MDC0, MDD0, and MDE0). These registers store the five (SHA-1) or four (MD5) 32-bit longwords that are the final answer (digest/context) of the hashing process. Message digest data may only be read if the MDSR[DONE] bit is set. Any reads prior to this result is an early read error (MDISR[ERE]). The message

digest registers always return all zeros when an error is generated. This register is cleared when the MDHA is reset or re-initialized. The reset values for the registers are the algorithms defined chaining variable values.

Address: 0xEC08_0030 (MDA0)   Reset: 0x0123_4567   Access: User read/write
         0xEC08_0034 (MDB0)   Reset: 0x89AB_CDEF
         0xEC08_0038 (MDC0)   Reset: 0xFEDC_BA98
         0xEC08_003C (MDD0)   Reset: 0x7654_3210
         0xEC08_0040 (MDE0)   Reset: 0xF0E1_D2C3

	31 30 29 28	27 26 25 24	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				MDA0, MDB0, MDC0, MDD0, MDE0				
W								
Reset				See Address Row				
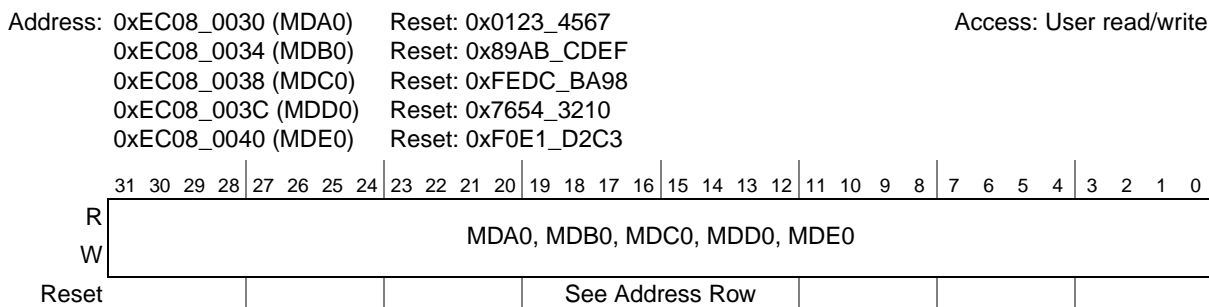
Figure 33-10. MDHA Message Digest Registers 0 (MD*x*0)

33.2.9 MDHA Message Data Size Register (MDMDS)

The MDMDS is a 32-bit register that, when read, stores the size of the current hash operation. This register is also used to write in the data size from a resumed hash operation. This data size is added to the MDDSR to complete the auto pad step.

Address: 0xEC08_0044 (MDMDS)   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				Message Data Size				
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
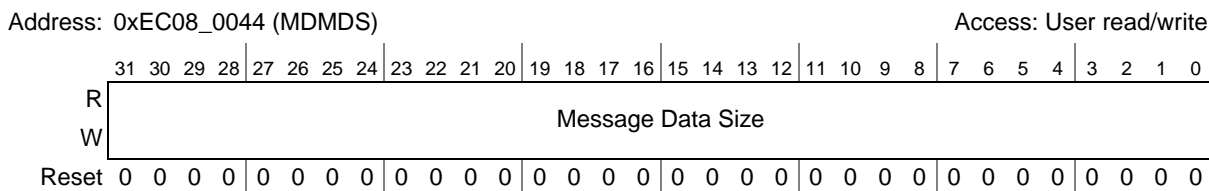
Figure 33-11. MDHA Message Data Size Register (MDMDS)

33.2.10 MDHA Message Digest Registers 1 (MD*x*1)

The MDHA message digest registers 1 consist of five 32-bit digest registers (MDA1, MDB1, MDC1, MDD1, and MDE1). These registers store the OPAD resulted digest to be used for the second hash operation in the HMAC or EHMAC mode. This digest is written directly to the message digest registers 0 after the first hash has been completed. The registers are write-only and any attempts to read from them always returns the value zero.

Address: 0xEC08_0070 (MDA1)   Access: User write-only
         0xEC08_0074 (MDB1)
         0xEC08_0078 (MDC1)
         0xEC08_007C (MDD1)
         0xEC08_0080 (MDE1)

	31 30 29 28	27 26 25 24	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 0
W				MDA1, MDB1, MDC1, MDD1, MDE1				
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
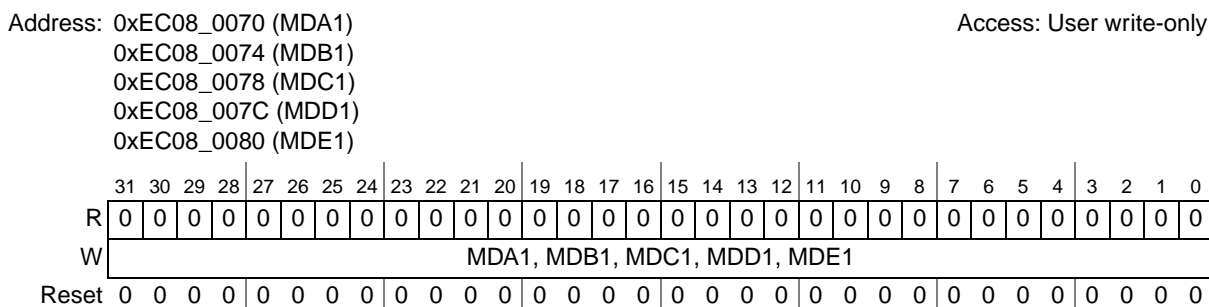
Figure 33-12. MDHA Message Digest Registers 1 (MD*x*1)

33.3 Functional Description

The MDHA module consists of three sub-blocks: the FIFO, MDHA top control block, and the MDHA logic block. A block level diagram of the MDHA module is shown in Figure 33-13.
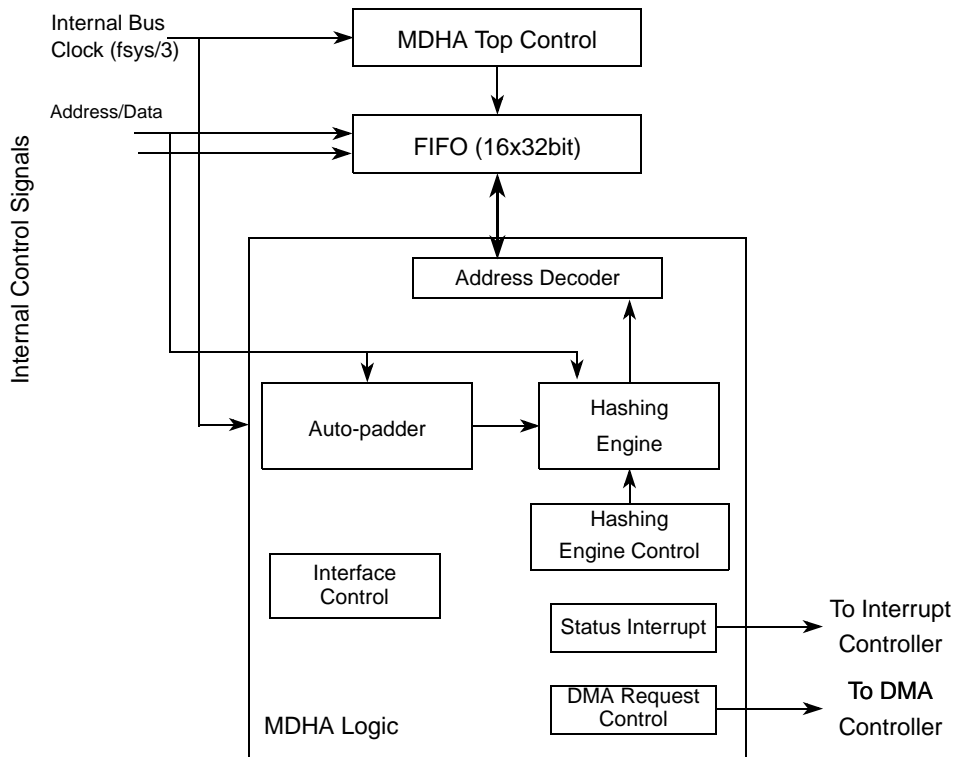
Figure 33-13. MDHA Block Diagram

33.3.1 MDHA Top Control

The MDHA Top Control block enables the FIFO whenever the MDHA module is enabled. This block also captures error and done status from the MDHA Logic block and generates a single interrupt to the interrupt controller.

33.3.2 FIFO

The FIFO block contains a 16×32-bit FIFO that is used for temporary storage of the data to be hashed.

33.3.3 MDHA Logic

The MDHA logic block consists of 7 sub-blocks: the address decoder, interface control, auto-padder, algorithm engine, algorithm engine control, and status interrupt as shown in Figure 33-13.

33.3.3.1 Address Decoder

The address decoder drives out the proper data from the module or captures incoming data into the appropriate register.

33.3.3.2 Interface Control

The interface block decodes the MDMR and outputs all control signals to all other blocks. Control signals are received from other modules to send a pop signal to the FIFO.

33.3.3.3 Auto-Padder

The auto-padder takes longwords in from the FIFO and then passes it directly to the engine or pads the word according to the control bits that are set. The IPAD and OPAD is done to all longwords in this block before they are passed directly to the engine. This block takes care of passing the proper data to the engine for the EHMAC mode of operation. This is done by an internal counter that leaves 351 bits in the input FIFO.

33.3.3.4 Hashing Engine

This module is the core of the Message Digest Hardware Accelerator that is capable of computing the Secure Hash Algorithm (SHA-1) or Message Digest 5 (MD5).

33.3.3.5 Hashing Engine Control

This module is the control unit of the MDHA that is capable of computing the Secure Hash Algorithm (SHA-1) and Message Digest 5 algorithm (MD5). This module keeps track of all rounds and tells the rest of the module when the operation has been completed.

33.3.3.6 DMA Request Control

This module is the control for the DMA request signal. It monitors the FIFO level and DMA request level and determines when the DMA request signal should be asserted and de-asserted.

33.3.3.7 Status Interrupt

This block generates the error interrupt if the host performs an illegal operation. The cause of the error is flagged in the MDISR (Section 33.2.5, "MDHA Interrupt Status & Mask Registers (MDISR and MDIMR)"). If an error occurs, the MDHA core engine is halted. This prevents the core from continuing operation with invalid data.

33.4 Initialization/Application Information

33.4.1 Performing a Standard HASH Operation

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)

3. MDMR register write. Select algorithm, data padding, and algorithm initialization.

4. Write message data into the FIFO in longwords.

5. MDDSR register write. Load this register with the length of the message data (without padding) in bytes.

6. Set MDCMR[GO].

7. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

NOTE

You must provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

8. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

33.4.2 Performing a Standard HASH Operation with DMA

1. Reset the MDHA using the MDCMR[SWR] bit.

2. MDCR register write. Enable the interrupts (optional), DMA, and set the DMA request level.

3. MDMR register write. Select algorithm, data padding, and algorithm initialization.

4. MDDSR register write. Load this register with the length of the message data (without padding) in bits.

5. The MDHA's DMA signalling makes the proper number of requests to the DMA controler to move data into the FIFO. The DMA controller must be programmed to acknowledge the MDHA's requests.

6. Set MDCMR[GO].

7. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

NOTE

You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

8. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

33.4.3 Performing a HMAC Operation Without the MACFULL Bit

The HMAC is done in three separate steps without the MACFULL bit. Each step requires the reinitialization of the MDHA.

33.4.3.1 Generation of Key with IPAD

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, IPAD, and MD initialization.
4. The data FIFO is filled with the key.
5. MDDSR register write. Load this register with the length of the key (without padding) in bytes.
6. MDHA does the required IPAD of key.
7. MDHA does the required algorithm's auto padding of message.
8. Set the MDCMR[GO] bit.
9. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

> **NOTE**
>
> You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

10. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest and the message digest count from the message digest registers.

33.4.3.2 Generation of Key with OPAD

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, OPAD, and MD initialization.
4. The data FIFO is filled with the key.
5. MDDSR register write. Load this register with the length of the key (without padding) in bytes.
6. MDHA does the required OPAD of key.
7. MDHA does the required algorithm's auto padding of message.
8. Set the MDCMR[GO] bit.
9. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

> **NOTE**
>
> You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

10. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest and the message digest count from the message digest registers.

33.4.3.3 HMAC Hash

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts. (optional)
3. MDMR register write. Select algorithm, data padding, and HMAC.
4. MDMDS register write. Load this register with the length of the message from the OPAD step.
5. Direct context load of (OPAD XOR key) into MDx1 registers from the OPAD step in the previous section.
6. Direct context load of (IPAD XOR key) into MDx0 registers from the IPAD step.
7. Direct context load of MDMDS register from the IPAD step (this should be 64 bytes).
8. Fill data FIFO with message to be hashed.
9. MDDSR register write. Load this register with the length of the message data (without padding) in bytes.
10. MDHA does the required algorithm's auto padding of the message.
11. Set the MDCMR[GO] bit.
12. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

NOTE

You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

13. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

33.4.4 Performing a SHA-1 EHMAC

EHMAC can only be performed with the SHA-1 algorithm. The Enhanced HMAC requires the keys to be hashed with IPAD and OPAD prior to the step below. The IPAD and OPAD step can be followed in Section 33.4.3.1, "Generation of Key with IPAD" and Section 33.4.3.2, "Generation of Key with OPAD."

1. Reset the MDHA using the MDCMR[SWR] bit.
2. MDCR register write. Enable the interrupts.
3. MDMR register write. Select algorithm, data padding, and EHMAC.
4. Direct context load of (OPAD XOR key) into MDx1 registers from OPAD step.
5. MDMDS register write. Load this register with the length of the message from the OPAD step.
6. Direct context load of (IPAD XOR key) into MDx0 registers from IPAD step.
7. Direct context load of MDMDS register from the IPAD step (this should be 64 bytes).
8. Fill data FIFO with message to be hashed.
9. MDDSR register write. Load this register with the length of the message data (without padding) in bits.

10. MDHA does the required algorithm's auto padding of the message.

11. Set the MDCMR[GO] bit.

12. Wait for MDSR[DONE] to be set or done interrupt to be triggered to indicate successful completion (or failure).

NOTE

You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

13. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

33.4.5   Performing a MAC Operation With the MACFULL Bit

The HMAC/EHMAC is done in one step with the MACFULL bit.

1. Reset the MDHA using the MDCMR[SWR] bit.

2. MDCR register write. Enable the interrupts. (optional)

3. MDMR register write. Select algorithm, data padding, HMAC or EHMAC, and MACFULL bits.

4. Direct context load of key into MDx1 registers.

5. MDMDS register write. Load this register with the length of the key.

6. Fill data FIFO with message to be hashed.

7. MDDSR register write. Load this register with the length of the message data (without padding) in bytes.

8. MDHA does the required algorithm's auto padding of the message.

9. Set the MDCMR[GO] bit.

10. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

NOTE

You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

11. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

33.4.6   Performing an NMAC

An NMAC consists of one Hash operation.

1. Reset the MDHA using the MDCMR[SWR] bit.

2. MDCR register write. Enable the interrupts. (optional)

3. MDMR register write. Select algorithm, data padding, HMAC.

4. Direct context load of the second key into DDx1 registers from OPAD step.

5. MDMDS register write. Load this register with the length of the key.

6. Direct context load of the first key into MDx0 registers from IPAD step.

7. MDDSR register write. Load this register with the length of the message data (without padding) in bits.

8. Direct context load of MDMDS register with key size (for NMAC load with data size of 0 bytes).

9. Fill data FIFO with message to be hashed.

10. MDHA does the required algorithm's auto padding of the message.

11. Set the MDCMR[GO] bit.

12. Wait for MDSR[INT] to be set or done interrupt to be triggered to indicate successful completion (or failure).

NOTE

You need to provide a time-out feature in the interrupt handler. The MDHA stalls with no response if it is waiting for message data. This most likely occurs if the MDDSR write is not received or auto-padding is disabled and a partial message block is provided.

13. If MDSR[DONE] is set or done interrupt is triggered, then read the message digest from the message digest registers.

Chapter 34
Random Number Generator (RNG)

34.1 Introduction

This chapter describes the random number generator (RNG), including a programming model, functional description, and application information.

NOTE

The MCF5327, MCF53281, and MCF5328 do not contain cryptography modules. Refer to Table 1-1 for details on device configurations.

34.1.1 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

34.2 Memory Map/Register Definition

Table 34-1 shows the address map for the RNG module. Detailed register descriptions are found in the following section.

Table 34-1. RNG Block Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC08_8000	RNG Control Register (RNGCR)	32	R/W	0x0000_0000	34.2.1/34-2
0xEC08_8004	RNG Status Register (RNGSR)	32	R	0x0010_0000	34.2.2/34-3
0xEC08_8008	RNG Entropy Register (RNGER)	32	W	0x0000_0000	34.2.3/34-4
0xEC08_800C	RNG Output FIFO (RNGOUT)	32	R	0x0000_0000	34.2.4/34-4

34.2.1 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.

Address: 0xEC08_8000 (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
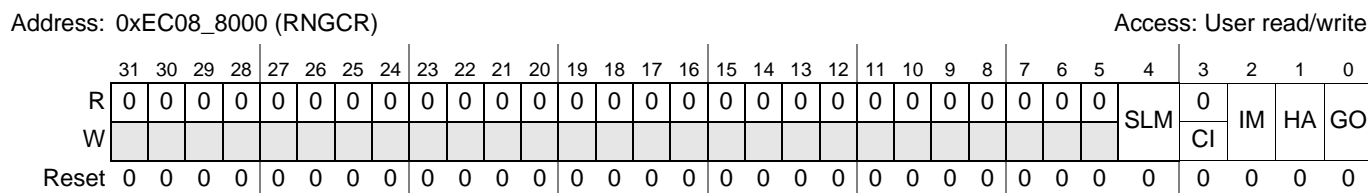
Figure 34-1. RNG Control Register (RNGCR)

Table 34-2. 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 34-2. 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.

34.2.2 RNG Status Register (RNGSR)

The RNGSR, shown in Figure 34-2, is a read only register which reflects the internal status of the RNG.

Address: 0xEC08_8004 (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 34-2. RNG Status Register (RNGSR)

Table 34-3. 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 34-3. 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.

34.2.3   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.

Address: 0xEC08_8008 (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
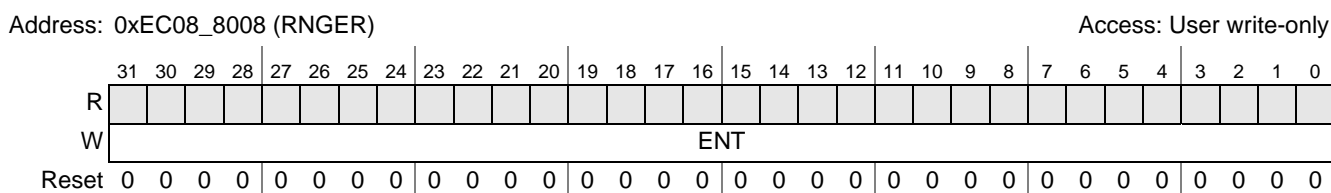
Figure 34-3. RNG Entropy Register (RNGER)

34.2.4   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 34.2.2, "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.

Address: 0xEC08_800C (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
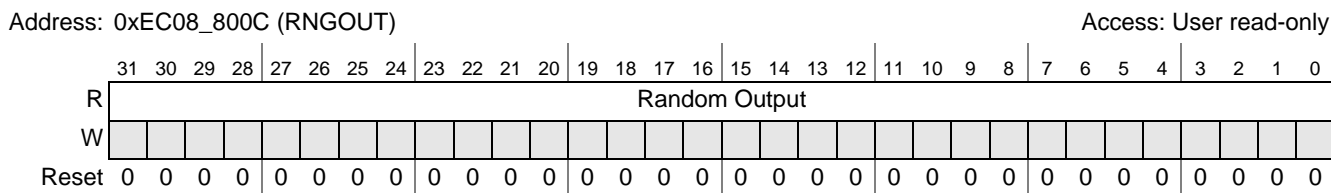
Figure 34-4. RNGOUT

34.3   Functional Description

Figure 34-5 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.
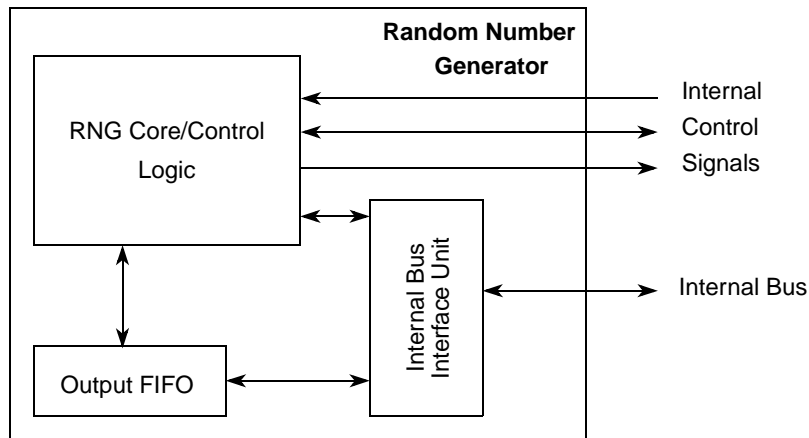
Figure 34-5. RNG Block Diagram

34.3.1   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.

34.3.2   RNG Core/Control Logic Block

This block contains the RNG's control logic as well as its core engine that generates random data.

34.3.2.1   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.

34.3.2.2   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.

34.4   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 35
Symmetric Key Hardware Accelerator (SKHA)

35.1 Introduction

The symmetric key hardware accelerator (SKHA) is a cryptographic hardware coprocessor designed to implement two widely used symmetric key block cipher algorithms, AES (Advanced Encryption Standard) and DES (Data Encryption Standard).

NOTE

The MCF5327, MCF53281, and MCF5328 do not contain cryptography modules. Refer to Table 1-1 for details on device configurations.

35.1.1 Features

The SKHA supports the following block ciphers:

- AES
    - 128-bit key
    - Electronic Code Book (ECB), Cipher Block Chaining (CBC), Counter (CTR) cipher modes.
- DES
    - 64 bit key (with parity)
    - ECB, CBC, and CTR modes
- Triple-DES (3DES)
    - 2 key & 3 key (128-bits & 192-bits with parity)
    - Key parity check
    - ECB, CBC, and CTR modes

35.1.1.1 Data Ecryption Standard (DES & 3DES) Algorithm

The SKHA is able to perform bulk data encryption/decryption in compliance with the Data Encryption Standard algorithm (ANSI x3.92, FIPS 46-2), as well as 3DES, which is an extension of the DES algorithm.

In DES mode, the SKHA operates by permuting 64-bit data blocks with a shared key and an initialization vector (IV). The SKHA supports two modes of IV operation: Electronic Code Book (ECB) and Cipher Block Chaining (CBC).

The processor supplies data to the SKHA, and the data is encrypted and subsequently made available to the processor via an output FIFO. The session key is input to the block prior to encryption.

DES is a block cipher that uses a 56-bit key (64 bits with CRC) to encrypt 64-bit blocks of data, one block at a time. A conceptual diagram of this process is shown in Figure 35-1. DES is a symmetric algorithm, so each of the two communicating parties share the same 64-bit key for encryption and decryption. DES processing begins after this shared session key is agreed upon. The text or binary message to be encrypted (typically called plaintext) is partitioned into *n* sets of 64-bit blocks. Each block is processed, in turn, by the DES engine, producing *n* sets of encrypted (ciphertext) blocks. These blocks may be transmitted to the other entity.

Decryption is handled in the reverse manner. The ciphertext blocks are processed one at a time by a DES module in the recipient's system. The same key is used, and the DES block manages the key processing internally so that the plaintext blocks are recovered.
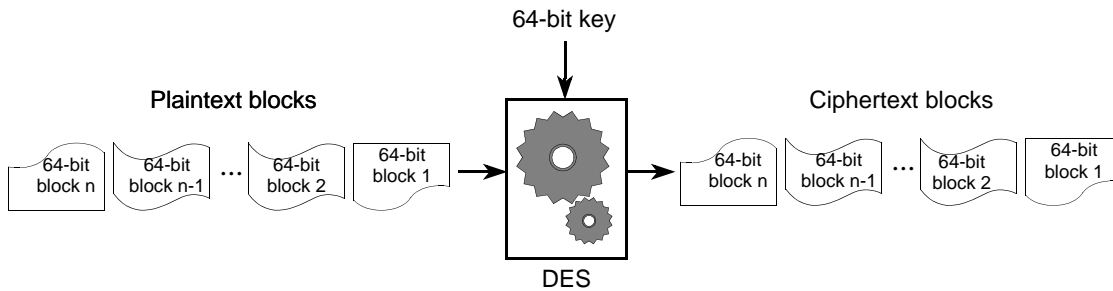
Figure 35-1. DES Encryption Process

In addition, the SKHA module can compute 3DES, which is an extension to the DES algorithm whereby every 64-bit input block is processed three times. The SKHA supports two key (K1=K3) or three key 3DES. A diagram of 3DES is shown in Figure 35-2.
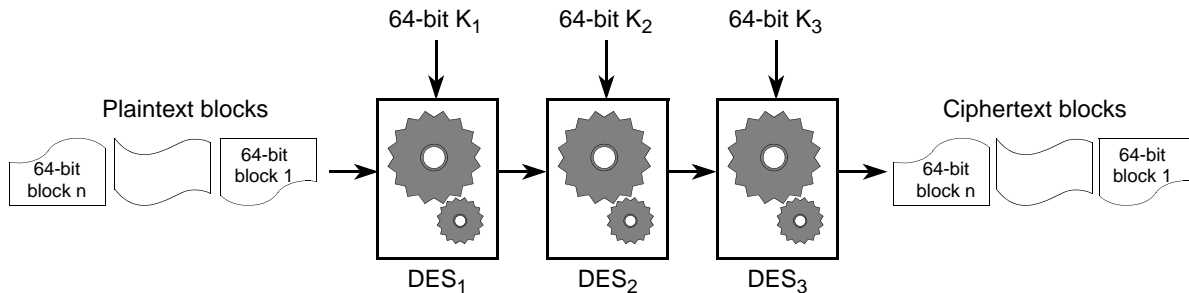
Figure 35-2. Triple-DES Encryption Process (ECB Mode)

35.1.1.2 Advanced Encryption Standard (AES) Algorithm

In AES mode, SKHA is used to accelerate bulk data encryption/decryption in compliance with the advanced encryption standard algorithm (AESA) Rinjdael. AES executes on 128-bit blocks with 128-bit key size.

AES is a symmetric key algorithm, the sender and receiver use the same key for encryption and decryption. The session key and initialization vector (CBC mode) are supplied to the SKHA module prior to

encryption. The processor supplies data to the module that is processed as 128-bit input. The SKHA engine performs ten rounds for encryption or decryption. AES operates in ECB, CBC, and CTR modes.
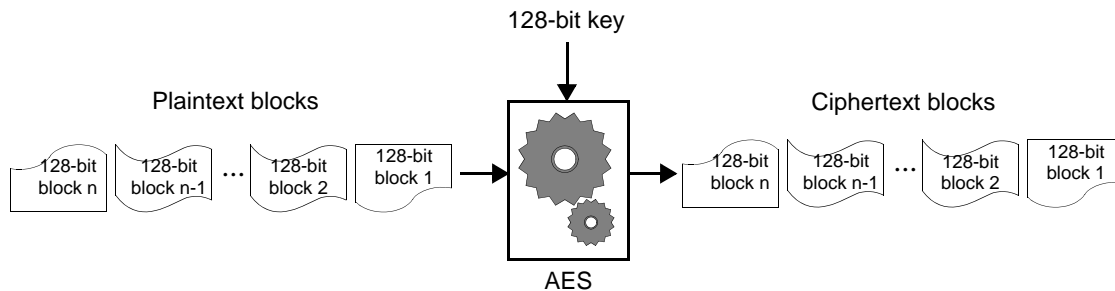
Figure 35-3. AES Encryption Process

35.1.1.3 Electronic Code Book (ECB) Cipher Mode

The simplest mode is ECB. Each block is processed independently, as shown in Figure 35-4. There is no context used in this mode. Each block of plaintext is encrypted to determine the corresponding ciphertext. In decrypt mode, the ciphertext blocks are decrypted to restore the plaintext.
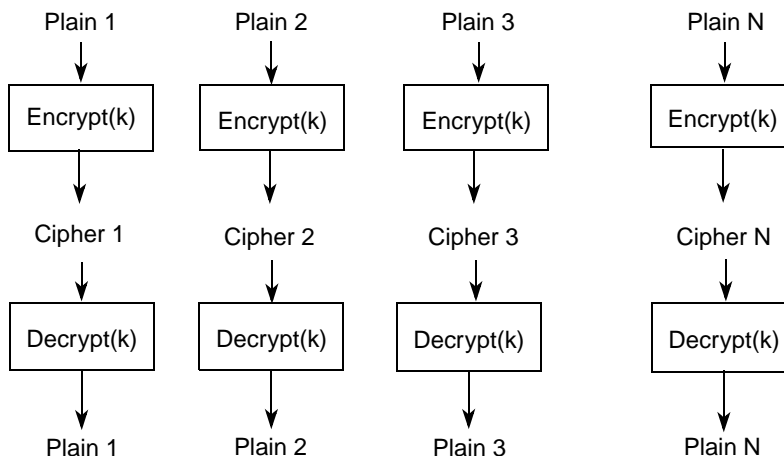
Figure 35-4. ECB Mode Processing

35.1.1.4 Cipher Block Chaining (CBC) Cipher Mode

Cipher Block Chaining mode encrypts the output of the previous block with the current block input as shown in Figure 35-5. For decryption, the previous ciphertext block is mixed with the decrypted block as shown in Figure 35-6. A 128-bit random initialization vector (IV) must be loaded prior to processing a message. The context registers are updated internally and must be read and restored during a context switch.
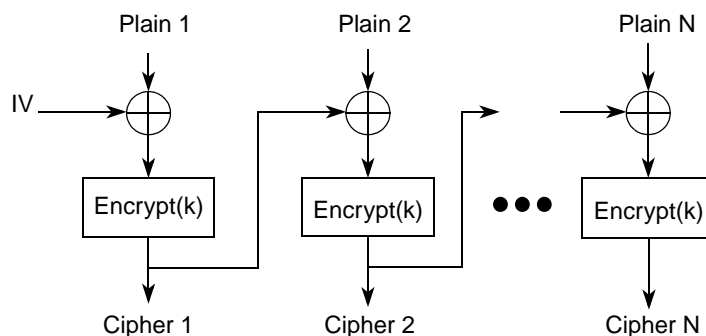
Figure 35-5. CBC Encryption
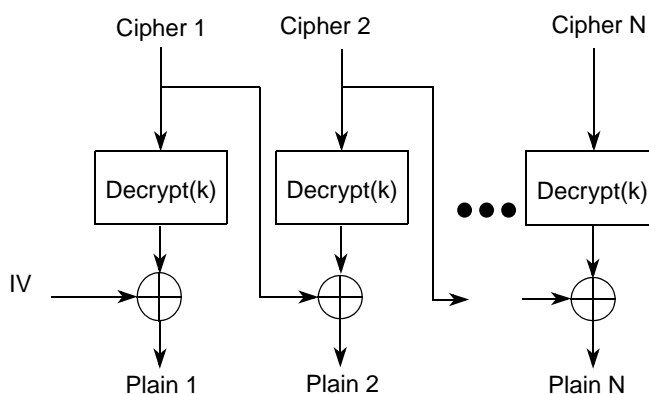
Figure 35-6. CBC Decryption

35.1.1.5 Counter (CTR) Cipher Mode

In counter mode, a random 128-bit initial counter value is incremented modulo 2^n with each block processed. The modulus size can be set between 2^8 through 2^{128}, by powers of 8 (SKMR[CTRM]). The running counter is encrypted and XOR'd with the plaintext to derive the ciphertext, or with the ciphertext to recover the plaintext as shown in Figure 35-7.
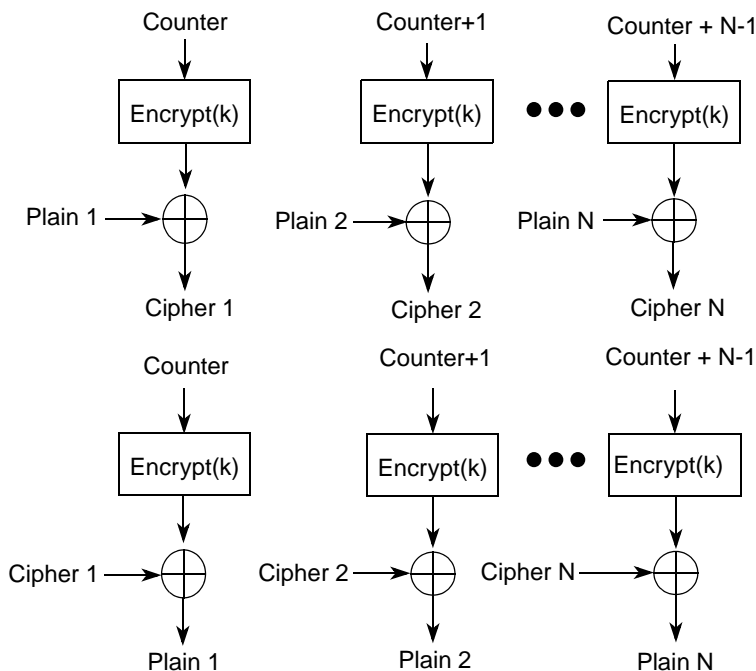
Figure 35-7. CTR Mode Processing

35.2 Memory Map/Register Definition

Table 35-1 shows the SKHA module memory map.

Table 35-1. SKHA Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC08_4000	SKHA Mode Register (SKMR)	32	R/W	0x0000_0000	35.2.1/35-6
0xEC08_4004	SKHA Control Register (SKCR)	32	R/W	0x0000_0000	35.2.2/35-7
0xEC08_4008	SKHA Command Register (SKCMR)	32	R/W	0x0000_0000	35.2.3/35-8
0xEC08_400C	SKHA Status Register (SKSR)	32	R	0x0000_0000	35.2.5/35-10
0xEC08_4010	SKHA Error Status Register (SKESR)	32	R	0x0000_0000	35.2.5/35-10
0xEC08_4014	SKHA Error Status Mask Register (SKEMR)	32	R/W	0x0000_0000	35.2.5/35-10
0xEC08_4018	SKHA Key Size Register (SKKSR)	32	R/W	0x0000_0000	35.2.6/35-12
0xEC08_401C	SKHA Data Size Register (SKDSR)	32	R/W	0x0000_0000	35.2.7/35-12
0xEC08_4020	SKHA Input FIFO (SKIN)	32	R/W	0x0000_0000	35.2.8/35-13
0xEC08_4024	SKHA Output FIFO (SKOUT)	32	R/W	0x0000_0000	35.2.9/35-13
0xEC08_4030	SKHA Key Data Register 1 (SKKDR1)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4034	SKHA Key Data Register 2 (SKKDR2)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4038	SKHA Key Data Register 3 (SKKDR3)	32	W	0x0000_0000	35.2.10/35-13

MCF5329 Reference Manual, Rev 3

Freescale Semiconductor · 35-5

Table 35-1. SKHA Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC08_403C	SKHA Key Data Register 4 (SKKDR4)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4040	SKHA Key Data Register 5 (SKKDR5)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4044	SKHA Key Data Register 6 (SKKDR6)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4070	SKHA Context 1 (SKC1)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4074	SKHA Context 2 (SKC2)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4078	SKHA Context 3 (SKC3)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_407C	SKHA Context 4 (SKC4)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4080	SKHA Context 5 (SKC5)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4084	SKHA Context 6 (SKC6)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4088	SKHA Context 7 (SKC7)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_408C	SKHA Context 8 (SKC8)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4090	SKHA Context 9 (SKC9)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4094	SKHA Context 10 (SKC10)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4098	SKHA Context 11 (SKC11)	32	R/W	0x0000_0000	35.2.11/35-14

35.2.1 SKHA Mode Register (SKMR)

The SKMR is used to select the cipher algorithm, direction, and cipher mode as shown in Figure 35-8. If the mode is modified during message processing, the SKESR[RM] bit is set. The mode may be modified after the module has completed processing, SKSR[DONE] is set.

NOTE

If reserved bits or undefined modes are set, a mode error is generated.

Address: 0xEC08_4000   Access: User read/write
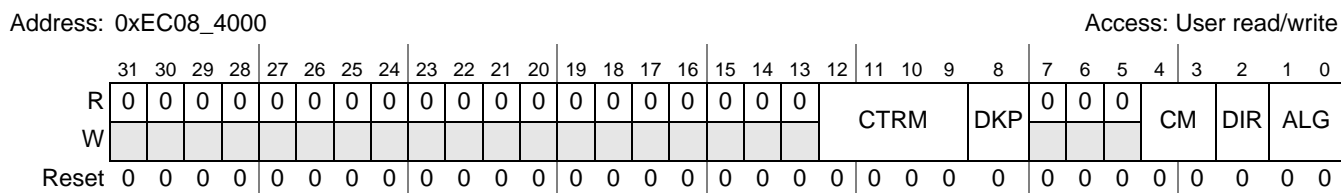
Figure 35-8. SKHA Mode Register (SKMR)

Table 35-2. SKMR Field Descriptions

Field	Description
31–13	Reserved, should be cleared.
12–9 CTRM	Counter mode modulus. Specifies modulus size for counter mode. In counter mode, the initial counter value is incremented modulo 2^N.

CTRM	CTR Modulus	
	DES or 3DES	AES
0000	2^8	2^8
0001	2^{16}	2^{16}
...	...	...
0110	2^{56}	2^{56}
0111	2^{64}	2^{64}
1000	Reserved[1]	2^{72}
1001	Reserved[1]	2^{80}
...	...	...
1110	Reserved[1]	2^{120}
1111	Reserved[1]	2^{128}

[1] If CTRM is set to reserved settings for DES or 3DES, a mode error occurs.

Field	Description
8 DKP	Disable key parity check. Disables checking DES parity. 0 Check for DES key parity errors 1 Do not check for DES key parity errors **Note:** A mode error is generated if this bit is set to one while in AES mode.
7–5	Reserved, should be cleared.
4–3 CM	Cipher mode. Selects the cipher mode. 00 ECB 01 CBC 10 Reserved 11 CTR
2 DIR	Direction. Selects encryption or decryption. 0 Decrypt 1 Encrypt
1–0 ALG	Algorithm. Selects which algorithm the SKHA module uses. 00 AES 01 DES 10 3DES 11 Reserved

35.2.2 SKHA Control Register (SKCR)

The SKCR contains bits that should be set after a hardware reset.

Address: 0xEC08_4004   Access: User read/write

Figure 35-9. SKHA Control Register (SKCR)

Table 35-3. SKMR Field Descriptions

Field	Description
31–30	Reserved, should be cleared.
20–24 ODMAL	Output DMA Request Level. Indicates the minimum number of words available in the output FIFO between DMA requests. This value must be 32 words or less. 000001–100000 = 1–32 words All others reserved.
23–22	Reserved, should be cleared.
21–16 IDMAL	Input DMA Request Level. Indicates the minimum number of words available in the input FIFO between DMA requests. This value must be 32 words or less. 000001–100000 = 1–32 words All others reserved.
15–4	Reserved, should be cleared.
3 END	Endian Select 0 Little endian mode 1 Big endian mode
2 ODMA	Output DMA Enable 0 Output DMA request signals disabled 1 Output DMA request signals enabled
1 IDMA	Input DMA Enable 0 Input DMA request signals disabled 1 Input DMA request signals enabled
0 IE	Interrupt enable. 0 Interrupts disabled 1 Interrupts enabled

35.2.3   SKHA Command Register (SKCMR)

Address 0xEC08_4008   Access: User read/write
:

Figure 35-10. SKHA Command Register (SKCMR)

Table 35-4. SKCMR Field Descriptions

Field	Description
31–4	Reserved, should be cleared.
3 GO	Go. Indicates that all data has been loaded into the module and the module should complete all processing. This bit is self resetting. 0 Do not finish processing 1 Complete all processing
2 CI	Clear Interrupt Request. Clears errors in the SKHA error status registers and deasserts any pending interrupt request to the interrupt controller. This bit is self resetting. 0 Do not clear interrupts & errors 1 Clear interrupt requests & errors
1 RI	Reinitialize. Reinitializes memory and clears all registers except SKHA Error Status Mask and Control registers. This bit is self clearing. 0 No Reinitialization 1 Reinitialize SKHA module
0 SWR	Software Reset. Functionally equivalent to a hardware resert. All registers are reset and FIFOs are cleared. This bit is self clearing. 0 No reset 1 Perform software reset

35.2.4 SKHA Status Register (SKSR)

The SKSR is read-only and reflects the current state of the SKHA. A write to this register has no effect.

Address: 0xEC08_400C   Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R				OFL								IFL				
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	BUSY	RD	ERR	DONE	INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
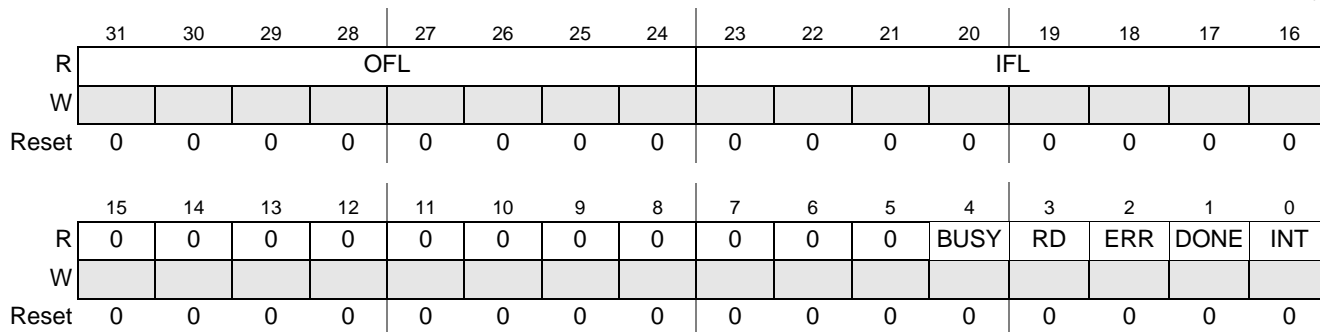
Figure 35-11. SKHA Status Register (SKSR)

Table 35-5. SKSR Field Descriptions

Field	Description
31–24 OFL	Output FIFO level. This 8-bit value indicates the number of data words in the Output FIFO.
23–16 IFL	Input FIFO level. This 8-bit value indicates the number data words in the Input FIFO.
15–5	Reserved, should be cleared.

Table 35-5. SKSR Field Descriptions (continued)

Field	Description
4 BUSY	Busy. Indicates the SKHA is busy. Mode, key data, context, and key size registers may not be modified and context registers may not be read while busy. 0 SKHA idle 1 SKHA busy
3 RD	Reset done. Indicates if reset of the SKHA module has completed. 0 Reset in progress 1 Reset complete
2 ERR	Error interrupt. Indicates that an error has occurred. 0 No error 1 Error occurred
1 DONE	Done interrupt. Indicates that the module has finished processing. 0 Not done 1 Done processing
0 INT	SKHA interrupt. Indicates that the module has finished processing data and the result is ready to be read from the Output FIFO or there is an error. This bit is set when an interrupt request in generated unless the SKHACR[IE] bit is cleared. 0 No interrupt 1 Done or error interrupt

35.2.5   SKHA Error Status and Mask Registers (SKESR, SKESMR)

The read-only SKESR indicates the type of error that has occurred. These errors are described below and shown in Table 35-6. When an error occurs, the SKHA engine halts and asserts an interrupt request to the interrupt controller. If multiple errors occur, only the first error is flagged. The SKHA must be reset when any error occurs. A write to this register has no effect.

Address: 0xEC08_4010   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	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	DRL	KRE	KPE	ERE	RMDP	KSE	DSE	IME	NEOF	NEIF	OFU	IFO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-12. SKHA Error Status Register (SKESR)

The SKESMR allows the user to mask off any of the SKESR bits. If the error occurs while the corresponding bit is set, the error is set in the SKESR but the interrupt is not generated. Additional errors are flagged in the SKESR until an unmasked error is generated.

NOTE

Masking errors should only be used for debug purposes. A masked error most likely causes invalid data.

MCF5329 Reference Manual, Rev 3

Address: 0xEC08_4014   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	DRL	KRE	KPE	ERE	RMDP	KSE	DSE	IME	NEOF	NEIF	OFU	IFO
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-13. SKHA Error Status Mask Register (SKESMR)

Table 35-6. SKESR & SKESMR Field Descriptions

Field	Description
31–12	Reserved, should be cleared.
11 DRL	DMA request level error. Indicates that the input and/or output DMA request level (SKMR[IDMAL] or SKMR[ODMAL]) was set to an invalid request size. When the IDMA or ODMA bit is set, the request level must be set to a value of 1-32 words. Zero is considered an invalid request level. 0 No error 1 Invalid DMA request level
10 KRE	Key read error. An illegal attempt to read the key registers during processing has been detected. 0 No error 1 Key read error occurred
9 KPE	Key parity error. Indicates if a DES key parity error has occurred. Key parity checking can be disabled by setting the SKMR[DKP] bit (See Section 35.2.1, "SKHA Mode Register (SKMR)"). 0 No error 1 Key parity error occurred
8 ERE	Early read. 0 No error 1 A context register was read from while the module was busy processing data.
7 RMDP	Register modified during processing 0 No error 1 A register was modified during processing
6 KSE	Key size error 0 No error 1 Illegal key size was written into the SKHA key size register
5 DSE	Data size error 0 No error 1 Illegal data size was written into the SKHA data size register
4 IME	Illegal mode error 0 No error 1 Illegal mode specified
3 NEOF	Non-empty output FIFO upon start. 0 No error 1 Output FIFO contains data upon start of processing

Table 35-6. SKESR & SKESMR Field Descriptions (continued)

Field	Description
2 NEIF	Non-empty input FIFO upon done. 0 No error 1 Input FIFO contained data when processing was complete
1 OFU	Output FIFO underflow 0 No error 1 Output FIFO was read while empty
0 IFO	Input FIFO overflow. 0 No error 1 Input FIFO has been written to while full

35.2.6 SKHA Key Size Register (SKKSR)

The 6-bit SKKSR is used to set the number of bytes in the key. For AES, this value is always 16 (0x10). For single DES, the key size must be 8 bytes (0x08). For two key Triple-DES the value should be 16 (0x10), while three key Triple-DES requires a key size of 24 bytes (0x18). The SKKSR must be set after the key data is written. After the key size is set, the contents of the key register are used to initialize the SKHA. If an illegal key size is specified, a key size error occurs, setting SKESR[KSE] and triggering an interrupt to the interrupt controller if KSE is not masked. If the key size is modified during message processing, a register modified error is generated, setting SKESR[RMDP] and triggering an interrupt to the interrupt controller if RMDP is not masked.

Address: 0xEC08_4018   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			Key Size in bytes			
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
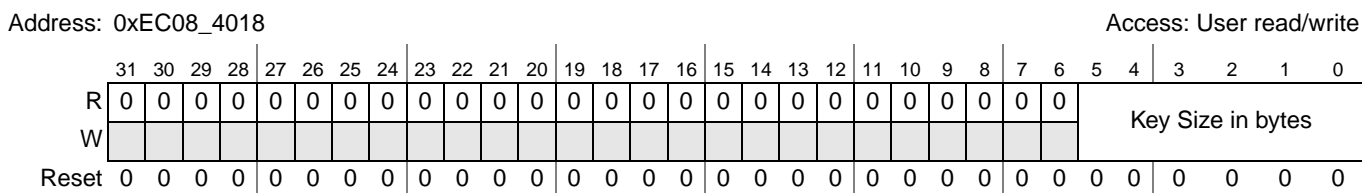
Figure 35-14. SKHA Key Size Register (SKKSR)

35.2.7 SKHA Data Size Register (SKDSR)

The 32-bit SKDSR holds the number of bytes to process. This value is set prior to loading message data. The SKHA internally decrements this value as it processes the message. The SKDSR may be read at any time to determine the number of bytes that remain to be processed. This value may be modified during message processing. The value written is internally added to the current value of data size. The SKHA continues to process data until the value in SKDSR reaches zero.

Address: 0xEC08_401C   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													Number of bytes to process																			
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
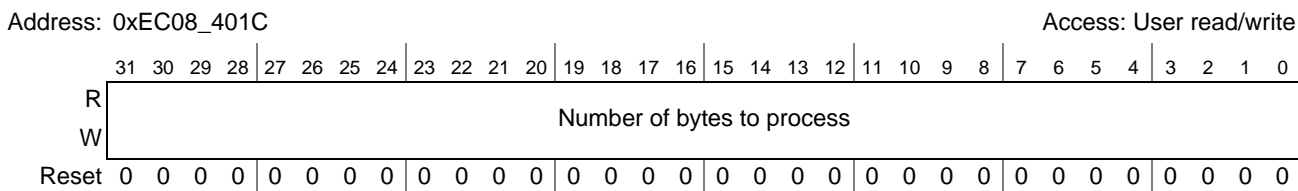
Figure 35-15. SKHA Key Size Register (SKKSR)

The value of data size must be a multiple of eight for DES/3DES or a multiple of 16 for AES. However, for CTR mode, data size may be a multiple of eight. If an illegal data size is set, a data size error is generated, setting SKESR[DSE] and generating an interrupt request to the interrupt controller if the DSE bit is not masked.

35.2.8 SKHA Input FIFO (SKIN)

The SKHA input FIFO provides temporary storage for data to be used during processing. The input FIFO is a write-only register and attempting to read from this register always returns zero. If the FIFO is written when the FIFO level is full, an interrupt is generated and the SKESR[IFO] bit is set. The SKSR[IFL] field, described in Section 35.2.4, "SKHA Status Register (SKSR)," can be polled to monitor how many 32-bit words are currently resident in the FIFO.

Address: 0xEC08_4020   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	0	0	0	0	0	0	0	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												Input FIFO																			
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
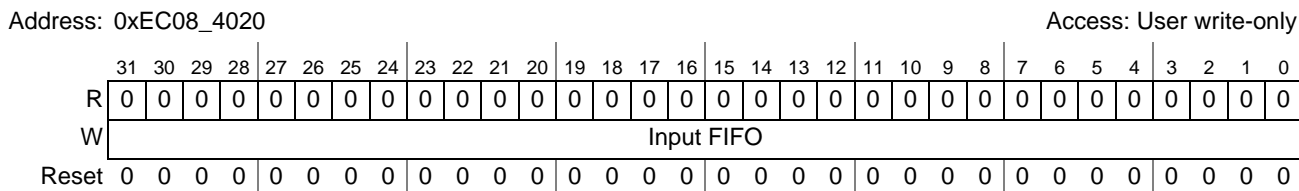
Figure 35-16. SKHA Input FIFO (SKIN)

35.2.9 SKHA Output FIFO (SKOUT)

The SKHA output FIFO provides temporary storage for data post-processing. The output FIFO is a read-only register and attempting to write to this register has no effect. If the output FIFO is read from when the FIFO level is empty then an interrupt is generated and SKESR[OFU] is set. The SKSR[OFL] field, described in Section 35.2.4, "SKHA Status Register (SKSR)," can be polled to monitor how many 32-bit words are currently resident in the FIFO.

Address: 0xEC08_4024   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												Output FIFO																			
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
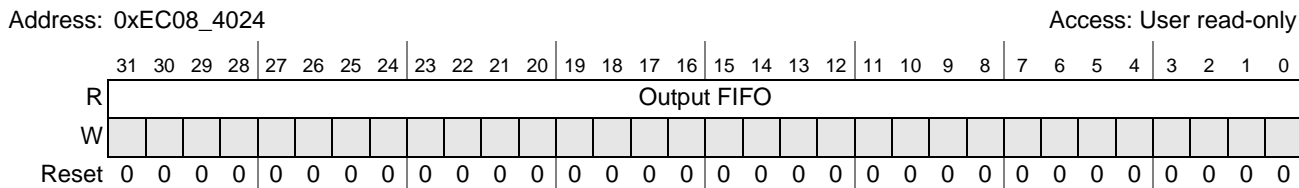
Figure 35-17. SKHA Output FIFO (SKOUT)

35.2.10 SKHA Key Data Registers (SKKDRn)

The six 32-bit SKKDRn are write-only and hold the symmetric key. The key should be loaded with the first four bytes placed in key 1, followed by key 2, etc. The key should be loaded before setting the key size. Any key data beyond the value specified in the SKKSR is ignored.

For AES, which uses a 128-bit key, SKKDR1–SKKDR4 must be written. For single DES, SKKDR1–SKKDR2 must be written (56-bit key & 8-bit parity). For 2-key 3DES, SKKDR1–4 must be written. For 3-key 3DES, SKKDR1–6 must be written. The SKKDRn registers are summarized in Table 35-7.

Table 35-7. SKHA Key Data Register Definitions

Algorithm	SKKDR1	SKKDR2	SKKDR3	SKKDR4	SKKDR5	SKKDR6
AES	Bytes 1–4	Bytes 5–8	Bytes 9–12	Bytes 13–16	—	—
DES	Bytes 1–4	Bytes 5–8	—	—	—	—
2 key 3DES	K1 Bytes 1–4	K1 Bytes 5–8	K2 Bytes 1–4	K2 Bytes 5–8	—	—
3 key 3DES	K1 Bytes 1–4	K1 Bytes 5–8	K2 Bytes 1–4	K2 Bytes 5–8	K3 Bytes 1–4	K3 Bytes 5–8

Attempting to read the SKKDR*n* registers generates an illegal address error. If the key registers are modified during message processing, a register modify error is generated, setting SKESR[RMDP] and triggering an interrupt to the interrupt controller (if RMDP is not masked).

Address: 0xEC08_4030 (SKKDR1)   Access: User write-only
0xEC08_4034 (SKKDR2)
0xEC08_4038 (SKKDR3)
0xEC08_403C (SKKDR4)
0xEC08_4040 (SKKDR5)
0xEC08_4044 (SKKDR6)
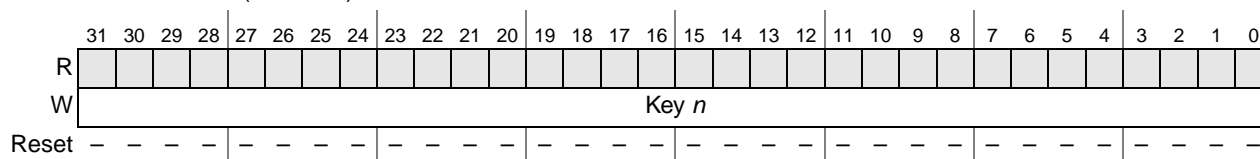
Figure 35-18. SKHA Key Data Registers (SKKDR*n*)

35.2.11 SKHA Context Registers (SKC*n*)

Prior to loading key data, the user must write the appropriate initialization data to the SKHA. Following a done interrupt, the user must read the contents of the SKC*n* registers prior to switching context. These values must be restored to resume processing of the original message. The SKC*n* registers are loaded differently depending on the algorithm and cipher mode. The location and values are summarized in Table 35-8. Context should be loaded with the lower bytes in the lowest 32-bit context register.

Example 35-1. Context Loading

```
0x0123456789ABCDEF10111213141516171819191A1B1C1D1E1F would be loaded as follows:
    SKC1 = 0x000102030405060708090A0B0C0D0E0F
    SKC2 = 0x10111213141516171819191A1B1C1D1E1F
```

Table 35-8. SKHA Context Register Definitions

Algorithm /Mode	SKHA Context Register *n* (32-bits each)											
	1	2	3	4	5	6	7	8	9	10	11	12
DES-ECB	—	—	—	—	—	—	—	—	—	—	—	—
DES-CBC	IV[1]	—	—	—	—	—	—	—	—	—	—	—

Table 35-8. SKHA Context Register Definitions

Algorithm /Mode	SKHA Context Register n (32-bits each)											
	1	2	3	4	5	6	7	8	9	10	11	12
AES-ECB	—	—	—	—	—	—	—	—	—	—	—	—
AES-CBC	IV[1]			—	—	—	—	—	—	—	—	—
DES-CTR	Counter[1]		—	—	—	—	—	—	—	—	—	—
AES-CTR	Counter[1]			—	—	—	—	—	—	—	—	—

[1] Must be written at start of new message

The value written to the IV/Nonce registers is replaced with the running initialization vector, IV, (CBC mode) or running counter (CTR mode) after processing begins. If any context registers are read prior to the SKSR[DONE] bit being set, an early read error, SKESR[ERE], is generated and an interrupt request is sent to the interrupt controller (if ERE is not masked).

Address: 0xEC08_4070 + 4×(n-1), where n=1–11 (SKCn) 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				Key n				
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
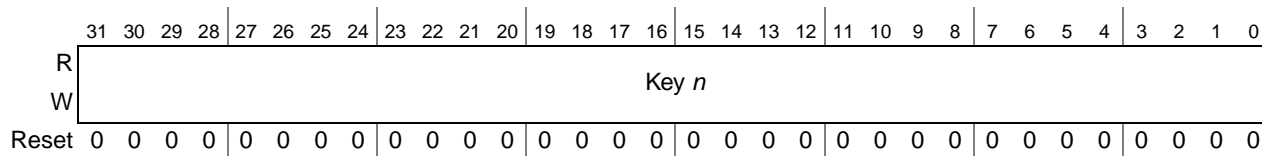
Figure 35-19. SKHA Context Registers (SKCn)

35.3 Functional Description

At the top level, the SKHA consists of seven functional blocks: The input FIFO, the output FIFO, transmit FIFO interface, receive FIFO interface, internal bus interface, top control and the SKHA logic.
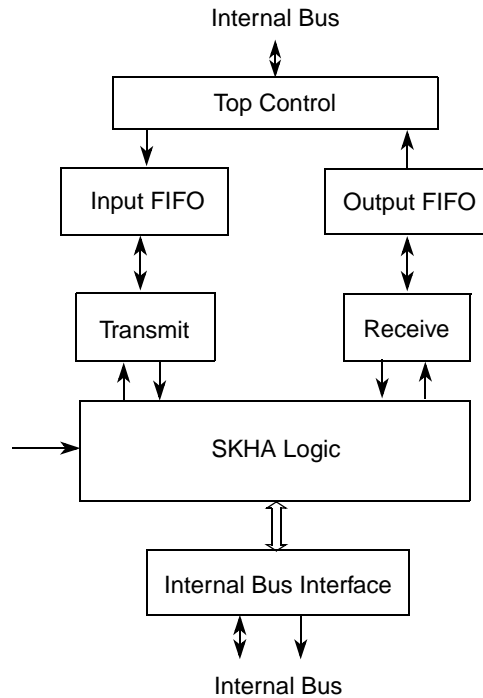
Figure 35-20. SKHA Block Diagram

35.3.1 Transmit FIFO Interface Block

This block translates the internal FIFO control signals to the input FIFO and passes the message data to the SKHA logic block. The SKHA logic block continues to pop data from the input FIFO until the FIFO is empty.

For AES, four 32-bit words are required to process a block. For DES/3DES, two 32-bit words are required to process a block. During the last word of the FIFO, the SKHA logic does not pop from the FIFO unless the SKCMR[GO] bit is set, indicating that the host has loaded all the message data.

35.3.2 Receive FIFO Interface Block

This block translates the internal FIFO control signals to the output FIFO and pass the processed message data from the SKHA logic block. The SKHA logic block pushes the same number of words to the output FIFO as it pops from the input FIFO. The SKHA logic block pushes to the output FIFO as long as the FIFO is not full. After the last word is pushed to the output FIFO, the done interrupt is asserted.

35.3.3 Top Control Block

This block generates the input and output FIFO transmit, receive and request signals and translates other internal signals at the top level.

35.3.4   SKHA Logic Block

This block contains the internal address decoder, addressable registers (key data, key size, data size, mode, context data, and status), interrupt and error logic, DMA input/output request control, and the core engine as shown in Figure 35-21.
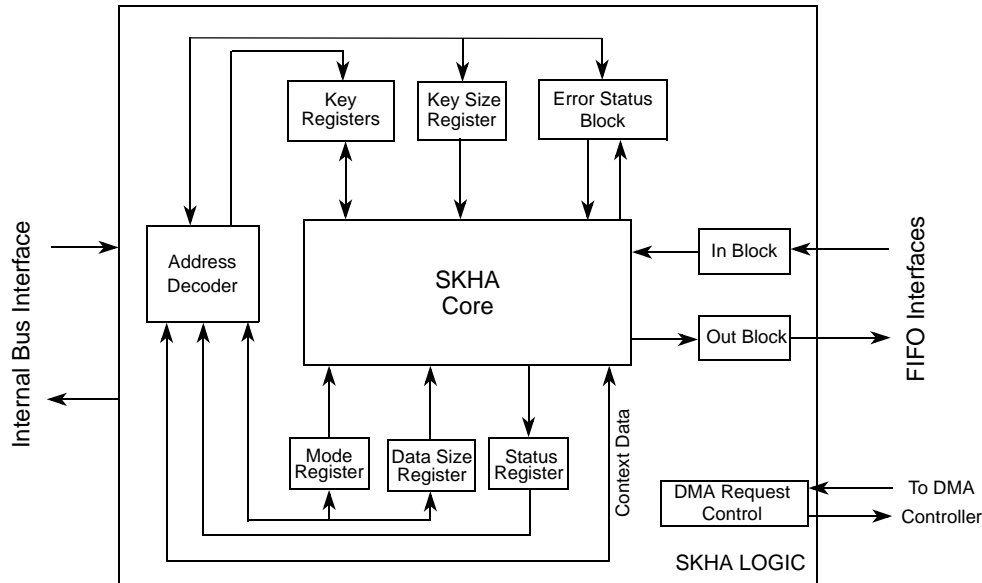
Figure 35-21. SKHA Logic Block Diagram

35.3.4.1   Address Decode Logic

The address decoder translates the internal address to write to the proper SKHA registers.

35.3.4.2   Error Interrupt/Status Logic

This block generates the error interrupt if the host performs an illegal operation. The cause of the error is flagged in the SKHA error status register (Section 35.2.5, "SKHA Error Status and Mask Registers (SKESR, SKESMR)") and an interrupt is triggered to the interrupt controller. If an error occurs, the SKHA core engine is halted. This prevents the core from continuing operation with invalid data. These error interrupts may be masked off selectively by setting the appropriate bits in the SKHA error status mask register (Section 35.2.5, "SKHA Error Status and Mask Registers (SKESR, SKESMR)")

35.3.4.3   DMA Request Control

This block is the control for the input and output DMA request signals. It monitors the input and output FIFO levels and DMA request levels (SKCR[ODMAL & IDMAL]) to determine when a DMA request should be triggered to the DMA controller.

35.3.4.4   SKHA Core

The heart of the SKHA is the core processing engine, Figure 35-22. The core contains the DES and AES block cipher engines. Also, the cipher mode (ECB, CBC, CTR) is implemented in this block. The SKHA logic block drives the cipher mode, algorithm, processing direction, key, and input block to the Mode Control logic.

While DES and AES operate differently internally, the Mode Control logic operates on top of the AES and DES engines. The Mode Control logic interfaces to both engines and feeds the input block and key to the selected engine. When the selected engine processes a block, it returns a "done" signal with the output block. The Mode Control logic in turn returns a "done" signal to the SKHA logic block along with the processed message block.

When the entire message is processed (following write to the End of Message register), the SKHA logic block sets SKSR[DONE] and generates an interrupt request to the interrupt controller. This indicates to that it is safe to read context.
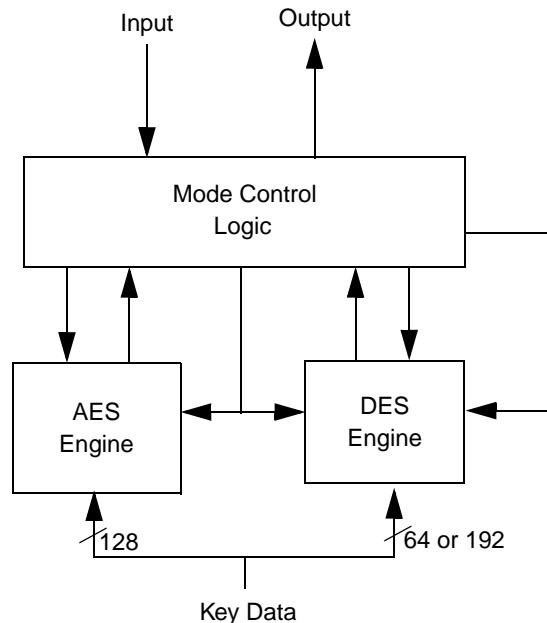
Figure 35-22. SKHA Core Block Diagram

35.3.5   Security Assurance Features

The SKHA features simple security assurance features to prevent operation in the presence of hardware faults and to shield visibility to sensitive data. Further, any user error or internal hardware fault causes the internal state machines to enter an idle/error state. In this case, the SKHA must be reset to resume operation, SKCMR[SWR]. Readable registers may be accessed to determine the nature of the error.

35.4 Initialization/Application Information

35.4.1 General Operation

The intended general operation of the SKHA is as follows for ECB, CBC and CTR modes:

1. Reset/initialize.
2. Set algorithm, processing direction, and cipher mode in the SKMR.
3. Write to SKCR to enable interrupts.
4. Write initialization vector/counter to the SKC*n* registers, if necessary.
5. Write symmetric key to SKKDR*n* registers.
6. Set number of bytes in key in the SKKSR.
7. Set the number of bits in the message in the SKDSR.
8. Load the input FIFO with message data.
9. Set the SKCMR[GO] bit.
10. Wait for interrupt, SKSR[INT].
11. Unload processed message data from the output FIFO.
12. Read contents of context registers, if necessary.
13. Set the SKCMR[CI] bit, to clear the done interrupt.

35.4.2 Operation with DMA

The intended DMA operation of the SKHA is as follows for ECB, CBC and CTR modes:

1. Reset/initialize.
2. Set algorithm, processing direction, and cipher mode in the SKMR.
3. Write to SKCR to enable interrupts, enable input and/or output DMA operation and set input and/or output DMA request levels.
4. Write initialization vector/counter to the SKC*n* registers, if necessary.
5. Write symmetric key to SKKDR*n* registers.
6. Set number of bytes in key in the SKKSR.
7. Set the number of bits in the message in the SKDSR.
8. Load the input FIFO with message data.
9. Set the SKCMR[GO] bit.
10. Wait for interrupt, SKSR[INT].
11. Unload processed message data from the output FIFO.
12. Read contents of context registers, if necessary.
13. Set the SKCMR[CI] bit, to clear the done interrupt.

35.4.3   Operation with Context Switch

The intended operation to process part of one message, followed by an intermediate message, and resume processing the original message is as follows:

1. Reset/initialize.
2. Set algorithm, set processing direction and cipher mode in the SKMR.
3. Write IV/Nonce to SKC*n* registers.
4. Write the symmetric key to SKKDR*n* registers.
5. Write the number of bytes in the key to SKKSR.
6. Set number of bits in the first part of message 1 in the SKDSR.
7. Load the input FIFO with first part of message 1.
8. Set the SKCMR[GO] bit.
9. Wait for done interrupt, SKSR[DONE].
10. Unload processed message data from the output FIFO.
11. Read context registers and store contents in memory.
12. Set the SKCMR[CI] bit, to clear the done interrupt.
13. Reset/initialize.
14. Process intermediate message as described in Section 35.4.1, "General Operation"
15. Reset/initialize.
16. Set algorithm, processing direction and cipher mode in the SKMR.
17. Restore context registers from memory.
18. Restore symmetric key to SKKDR*n*.
19. Write number of bytes in the key to the SKKSR.
20. Set number of bits in remaining part of message 1 to SKDSR.
21. Load the input FIFO with the remaining portion of message 1.
22. Set the SKCMR[GO] bit.
23. Wait for interrupt, SKSR[INT].
24. Unload processed message data from the output FIFO.

Chapter 36
Debug Module

36.1 Introduction

This chapter describes the revision B+ enhanced hardware debug module.

36.1.1 Block Diagram

The debug module is shown in Figure 36-1.
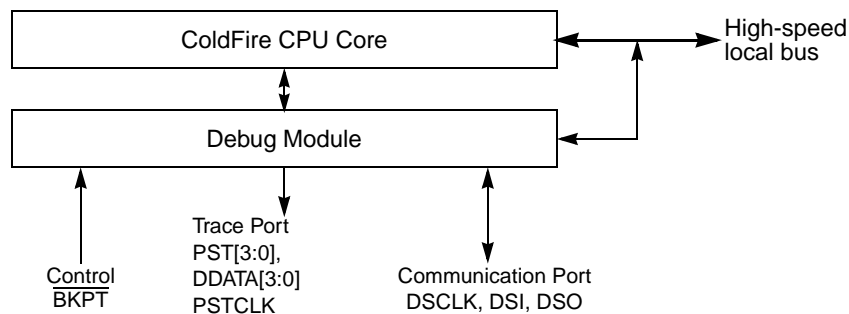
Figure 36-1. Processor/Debug Module Interface

36.1.2 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 36.4.4, "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 36.4.1, "Background Debug Mode (BDM)," and Section 36.3, "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 36.4.2, "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 36.3.2, "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 new PC breakpoint registers. For revision B, CSR[HRL] is 1, and for revision B+, CSR[HRL] is 0x9.

The following table summarizes the various debug revisions.

Table 36-1. 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) 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 new PC breakpoint registers PBR1–3
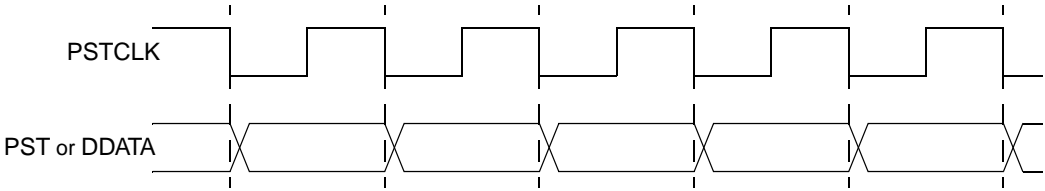
36.2 Signal Descriptions

Table 36-2 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 36.4.7, "Freescale-Recommended BDM Pinout".

Table 36-2. 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 (BKPT)	Input requests a manual breakpoint. Assertion of BKPT puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status signals (PST[3:0]) as the value 0xF. If CSR[BKD] is set (disabling normal BKPT functionality), asserting BKPT generates a debug interrupt exception in the processor.

Table 36-2. Debug Module Signals (continued)

Signal	Description
Processor Status Clock (PSTCLK)	Delayed version of the processor clock. Its rising edge appears in the center of valid PST and DDATA output. PSTCLK indicates when the development system should sample PST and DDATA values. The following figure shows PSTCLK timing with respect to PSTD and DATA. If real-time trace is not used, setting CSR[PCD] keeps PSTCLK, PST and DDATA 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 PST and DDATA outputs. PSTCLK starts clocking only when the first non-zero PST value (0xC, 0xD, or 0xF) occurs during system reset exception processing. Table 36-24 describes PST values.
Debug Data (DDATA[3:0])	These output signals display the register breakpoint status as a default, or optionally, captured address and operand values. The capturing of data values is controlled by the setting of the CSR. Additionally, execution of the WDDATA instruction by the processor captures operands that are displayed on DDATA. These signals are updated each processor cycle.
Processor Status (PST[3:0])	These output signals report the processor status. Table 36-24 shows the encoding of these signals. These outputs indicate the current status of the processor pipeline and, as a result, are not related to the current bus transfer. The PST value is updated each processor cycle.

36.3   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 36-3, 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 36.4.1.5, "BDM Command Set". These commands contain a 5-bit field, DRc, that specifies the register, as shown in Table 36-3.

<center>**Table 36-3. Debug Module Memory Map**</center>

DRc[4–0]	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W See Note	0x0098_0000	36.3.2/36-5
0x05	BDM address attribute register (BAAR)	32[1]	W	0x05	36.3.3/36-8
0x06	Address attribute trigger register (AATR)	32[1]	W	0x0005	36.3.4/36-9
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	36.3.5/36-10
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined	36.3.6/36-13
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined	36.3.6/36-13
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined	36.3.7/36-15
0x0D	Address breakpoint low register (ABLR)	32	W	Undefined	36.3.7/36-15
0x0E	Data breakpoint register (DBR)	32	W	Undefined	36.3.8/36-16
0x0F	Data breakpoint mask register (DBMR)	32	W	Undefined	36.3.8/36-16
0x18	PC breakpoint register 1 (PBR1)	32	W	See Section	36.3.6/36-13
0x1A	PC breakpoint register 2 (PBR2)	32	W	See Section	36.3.6/36-13
0x1B	PC breakpoint register 3 (PBR3)	32	W	See Section	36.3.6/36-13

[1] Each debug register is accessed as a 32-bit register; reserved fields are not used (don't care).

<center>**NOTE**</center>

> 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.

36.3.1   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 36-4.

Table 36-4. 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.

36.3.2 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
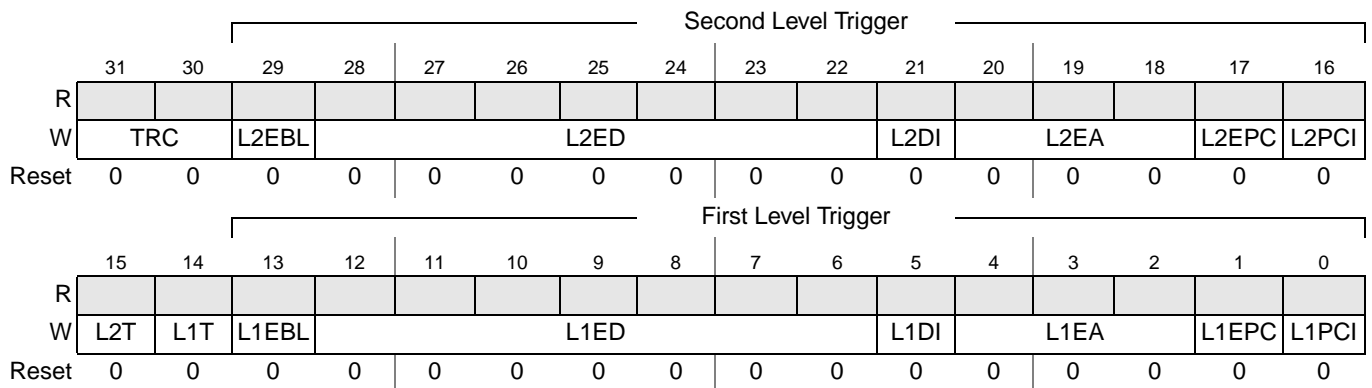   BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	BSTAT				FOF	TRG	HALT	BKPT	HRL				DBT	BKD	PCD	IPW
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	1	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	0	SPD	FDBG	DBGH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-2. Configuration/Status Register (CSR)

Table 36-5. CSR Field Descriptions

Field	Description
31–28 BSTAT	Breakpoint Status. Provides read-only status (from the BDM port only) information concerning hardware breakpoints. BSTAT is cleared by a TDR 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
27 FOF	Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM. FOF is cleared when CSR is read (from the BDM port only).
26 TRG	Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command or reading CSR (from the BDM port only) clear TRG.
25 HALT	Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clear HALT.
24 BKPT	Breakpoint assert. If BKPT is set, $\overline{BKPT}$ was asserted, forcing the processor into BDM. Reset, the debug GO command, or reading CSR (from the BDM port only) clear BKPT.
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+ (This is the value used for this device) 1011 Revision D+
19 DBT	Debug translate. Indicates, from the BDM port only, the presence of the debug translate block that compresses the PST/DDATA stream to 1/2 the core clock. 0 Debug translate block is not present 1 Debug translate block is present
18 BKD	Breakpoint disable. Disables the normal $\overline{BKPT}$ input signal functionality, and allows the assertion of this pin to generate a debug interrupt. 0 Normal operation 1 $\overline{BKPT}$ is edge-sensitive: a high-to-low edge on $\overline{BKPT}$ 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 **Note:** When PCD is set, do not execute a wddata instruction or perform any debug captures. Doing so, hangs the device.
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.

Table 36-5. CSR Field Descriptions (continued)

Field	Description
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 36.4.2.2, "Emulator Mode".
12–11 DDC	Debug data control. Controls operand data capture for DDATA, 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 36-24. 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.
9–8 BTB	Branch target bytes. Defines the number of bytes of branch target address DDATA 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 36.4.4.1, "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. 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.

Table 36-5. CSR Field Descriptions (continued)

Field	Description
3	Reserved, must be cleared.
2 SPD	PST/DDATA speed. Controls whether PST/DDATA is processed by the debug translate block that compresses the data to 1/2 the core frequency. 0 Half speed 1 Full speed
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.

36.3.3 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
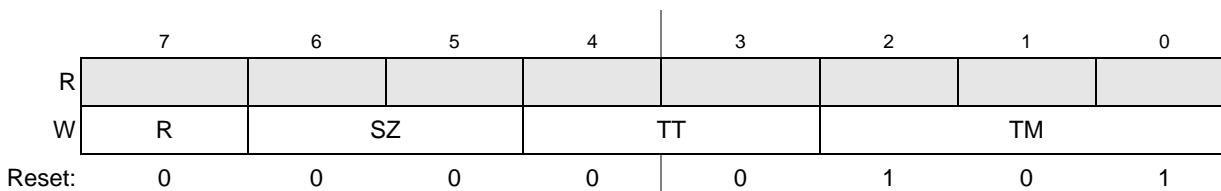
Figure 36-3. BDM Address Attribute Register (BAAR)

Table 36-6. 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

Table 36-6. BAAR Field Descriptions (continued)

Field	Description
4–3 TT	Transfer Type. See the TT definition in the AATR description, Section 36.3.4, "Address Attribute Trigger Register (AATR)".
2–0 TM	Transfer Modifier. See the TM definition in the AATR description, Section 36.3.4, "Address Attribute Trigger Register (AATR)".

36.3.4   Address Attribute Trigger Register (AATR)

The AATR defines 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). AATR is accessible in supervisor mode as debug control register 0x06 using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x06 (AATR)   Access: Supervisor write-only
                                                                                  BDM write-only

	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
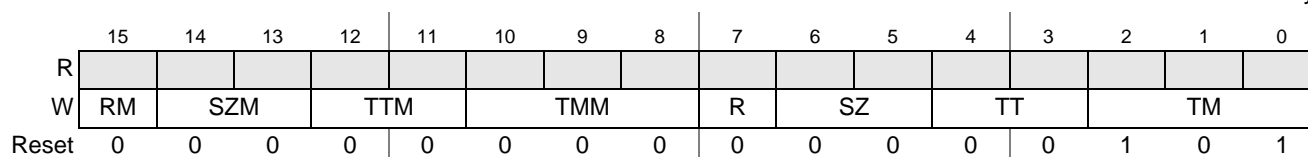
Figure 36-4. Address Attribute Trigger Register (AATR)

Table 36-7. AATR Field Descriptions

Field	Description
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 36-7. AATR Field Descriptions (continued)

Field	Description
4–3 TT	Transfer Type. Compared with the local bus transfer type signals. 00 Normal processor access 01 Reserved 10 Emulator mode access 11 Reserved 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.
2–0 TM	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).

TM	TT=00 (normal mode)	TT=10 (emulator mode)
000	Explicit cache line push	Reserved
001	User data access	Reserved
010	User code access	Reserved
011	Reserved	Reserved
100	Reserved	Reserved
101	Supervisor data access	Emulator mode access
110	Supervisor code access	Emulator code access
111	Reserved	Reserved

36.3.5   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. Breakpoint logic may be configured as a one- or two-level trigger. TDR[31–16] bits define second-level trigger, and bits 15–0 define first-level trigger.

NOTE

The debug module has no hardware interlocks to prevent spurious breakpoint triggers while the breakpoint registers are being loaded. Disable TDR (by clearing TDR[29,13]) before defining triggers.

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.

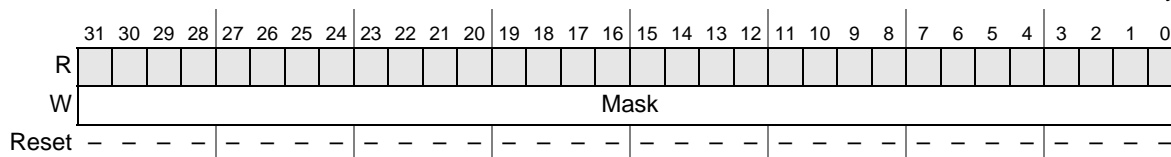DRc[4:0]: 0x07 (TDR) Access: Supervisor write-only
BDM write-only

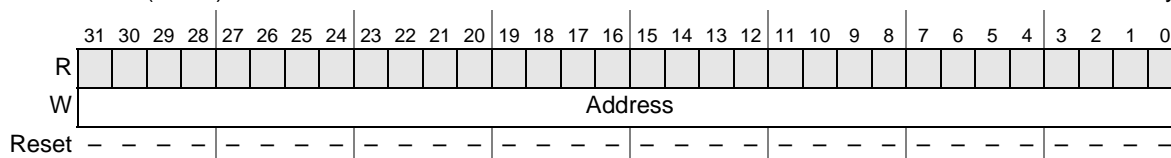Second Level Trigger

	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	L2T	L1T	L1EBL				L1ED				L1DI		L1EA		L1EPC	L1PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-5. Trigger Definition Register (TDR)

Table 36-8. TDR Field Descriptions

Field	Description
31–30 TRC	Trigger Response Control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on DDATA. 00 Display on DDATA only 01 Processor halt 10 Debug interrupt 11 Reserved
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. <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>
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 36-8. TDR Field Descriptions (continued)

Field	Description
20–18 L2EA	Enable Level 2 Address Breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint. | TDR Bit	Description | |---|---| | 20 | Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR. | | 19 | Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR. | | 18 | Address breakpoint low. The breakpoint is based on the address in the ABLR. |
17 L2EPC	Enable Level 2 PC Breakpoint. 0 Disable PC breakpoint 1 Enable PC breakpoint where the trigger is defined by the logical summation of: $$(\text{PBR0 and } \overline{\text{PBMR}}) \mid \text{PBR1} \mid \text{PBR2} \mid \text{PBR3} \qquad \textit{Eqn. 36-1}$$
16 L2PCI	Level 2 PC Breakpoint Invert. 0 The PC breakpoint is defined within the region defined by PBR*n* and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR*n* and PBMR.
15 L2T	Level 2 Trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range & Data_condition) where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers. 0 Level 2 trigger = PC_condition & Address_range & Data_condition 1 Level 2 trigger = PC_condition | (Address_range & Data_condition) **Note:** Debug Rev A only had the AND condition available for the triggers.
14 L1T	Level 1 Trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range & Data_condition) where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers. 0 Level 1 trigger = PC_condition & Address_range & Data_condition 1 Level 1 trigger = PC_condition | (Address_range & Data_condition) **Note:** Debug Rev A only had the AND condition available for the triggers.
13 L1EBL	Enable Level 1 Breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 1 breakpoints 1 Enables all level 1 breakpoint triggers

Table 36-8. TDR Field Descriptions (continued)

Field	Description
12–6 L1ED	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>
5 L1DI	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.
4–2 L1EA	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>
1 L1EPC	Enable Level 1 PC breakpoint. 0 Disable PC breakpoint 1 Enable PC breakpoint
0 L1PCI	Level 1 PC Breakpoint Invert. 0 The PC breakpoint is defined within the region defined by PBR*n* and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR*n* and PBMR.

36.3.6   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 is 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. 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 36.4.1.5, "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	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
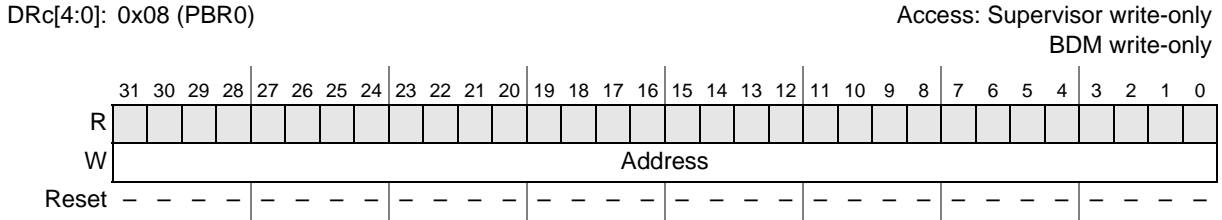
Figure 36-6. PC Breakpoint Register (PBR0)

Table 36-9. 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)   Access: Supervisor write-only
0x1A (PBR2)   BDM write-only
0x1B (PBR3)

	31	30	29	28	27	26	25	24	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
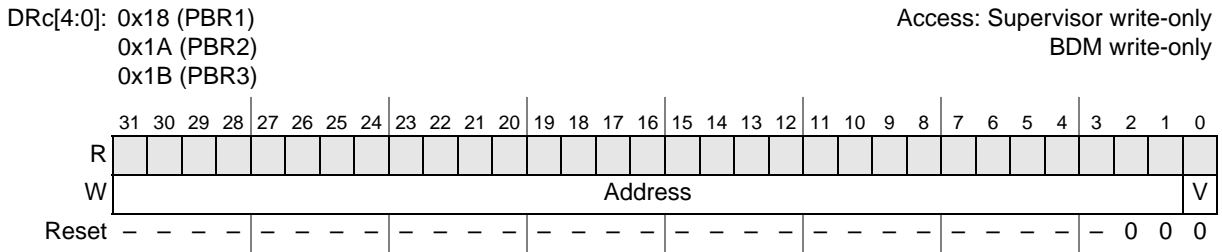
Figure 36-7. PC Breakpoint Register *n* (PBR*n*)

Table 36-10. 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 36-8 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 36-8. PC Breakpoint Mask Register (PBMR)

Table 36-11. 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.

36.3.7 Address Breakpoint Registers (ABLR, ABHR)

The ABLR and ABHR 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

ABLR and ABHR are accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command.

DRc[4:0]: 0x0C (ABHR)
        0x0D (ABLR)

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 36-9. Address Breakpoint Registers (ABLR, ABHR,)

Table 36-12. ABLR 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.

Table 36-13. ABHR Field Description

Field	Description
31–0 Address	High Address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

36.3.8 Data Breakpoint and Mask Registers (DBR, DBMR)

The data breakpoint register (DBR), specify data patterns used as part of the trigger into debug mode. DBR bits are masked by setting corresponding DBMR bits, as defined in TDR.

DBR and DBMR are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x0E (DBR)   Access: Supervisor write-only
                                                                                   BDM write-only
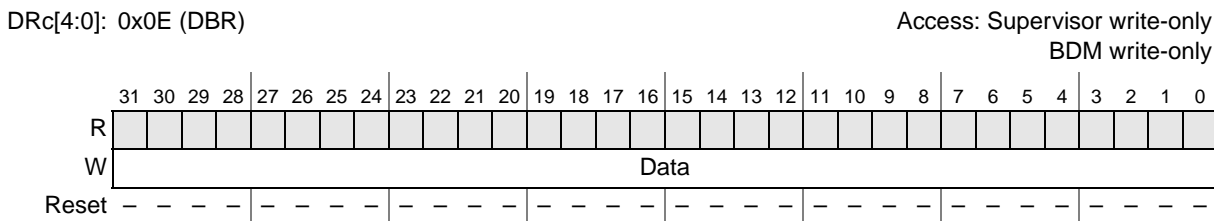
Figure 36-10. Data Breakpoint Registers (DBR)

Table 36-14. DBR 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)   Access: Supervisor write-only
                                                                                   BDM write-only
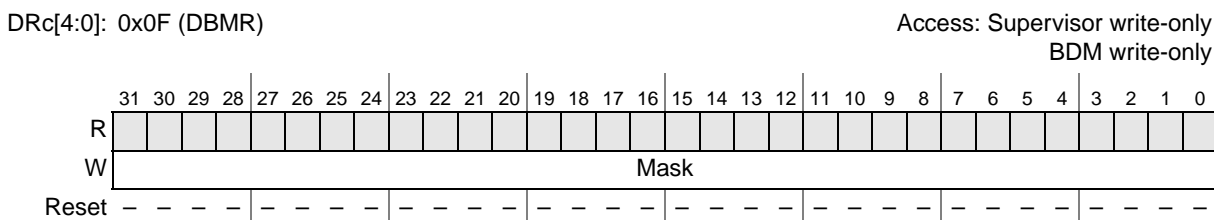
Figure 36-11. Data Breakpoint Mask Registers (DBMR)

Table 36-15. DBMR 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 DBR supports aligned and misaligned references. shows relationships between processor address, access size, and location within the 32-bit data bus.

Table 36-16. 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]

36.4 Functional Description

36.4.1 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 feature allows quick hardware debugging with the same tool set used for firmware development.

36.4.1.1 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 pocessor, 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 36.4.2.1, "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 $\overline{BKPT}$ input is treated as a pseudo-interrupt; asserting $\overline{BKPT}$ 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 $\overline{BKPT}$:

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the $\overline{BKPT}$ input is asserted within eight cycles after $\overline{RESET}$ is negated, the processor enters the halt state, signaling halt status (0xF) on the PST 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 $\overline{BKPT}$ 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.

36.4.1.2   BDM Serial Interface

When the CPU is halted and PST 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 36-2. 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 36-12, all state transitions are enabled on a rising edge of the PSTCLK clock when DSCLK is high; DSI is sampled and DSO is driven.
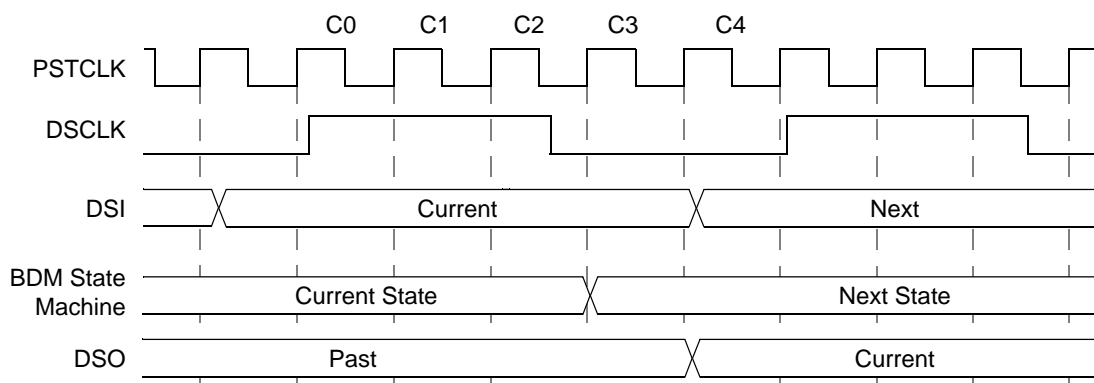
Figure 36-12. Maximum BDM Serial Interface Timing

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.

36.4.1.3 Receive Packet Format

The basic receive packet consists of 16 data bits and 1 status bit
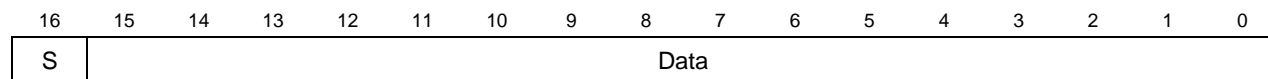
Figure 36-13. Receive BDM Packet

Table 36-17. Receive BDM Packet Field Description

Field	Description
16 S	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>
15–0 Data	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.

36.4.1.3.1 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								
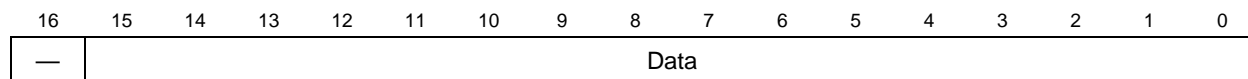
Figure 36-14. Transmit BDM Packet

Table 36-18. 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.

36.4.1.3.2 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)															
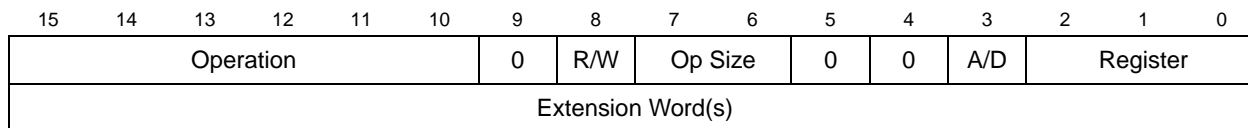
Figure 36-15. BDM Command Format

Table 36-19. BDM Field Descriptions

Field	Description
15–10 Operation	Specifies the command. These values are listed in Table 36-20.
9	Reserved, must be cleared.
8 R/W	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.
7–6 Op Size	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>
5–4	Reserved, must be cleared.
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 36-21.

36.4.1.3.3   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.

36.4.1.4   Command Sequence Diagrams

The command sequence diagram in Figure 36-16 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.
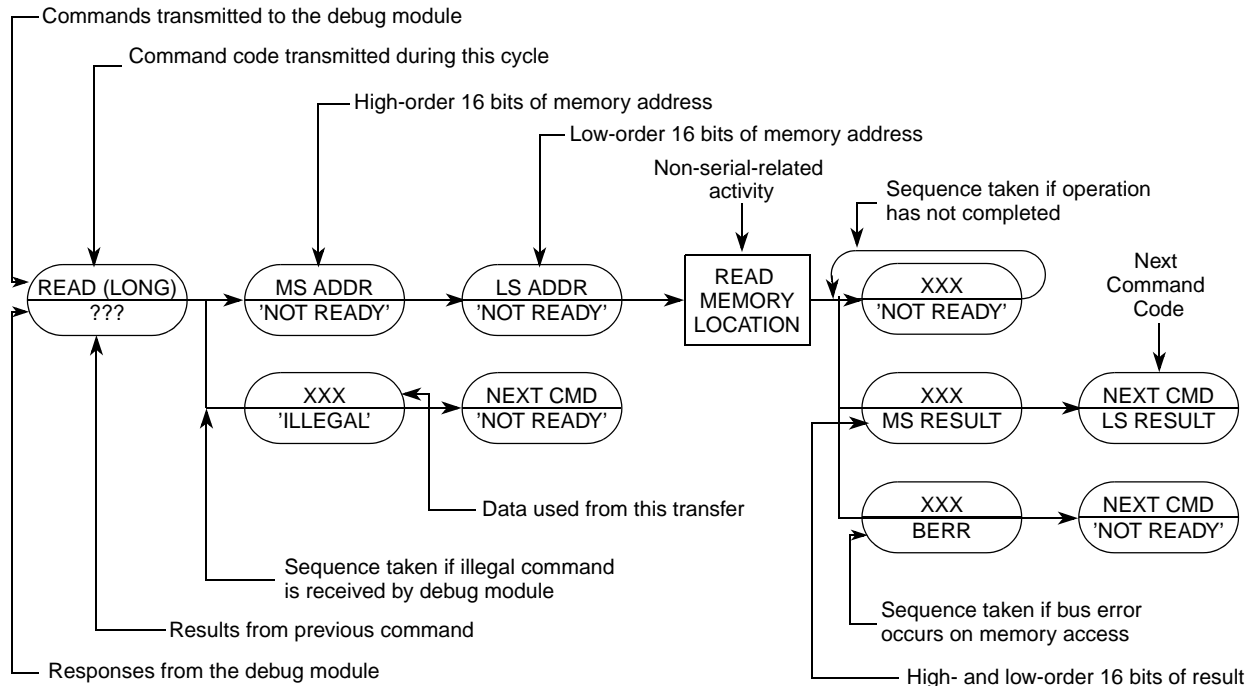
Figure 36-16. Command Sequence Diagram

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.

36.4.1.5   BDM Command Set

Table 36-20 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 36-21 for register address encodings.

Table 36-20. 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	36.4.1.5.1/36-24	0x218 {A/D, Reg[2:0]}
Write A/D register	WAREG/ WDREG	Write the data operand to the specified address or data register.	Halted	36.4.1.5.2/36-24	0x208 {A/D, Reg[2:0]}
Read memory location	READ	Read the data at the memory location specified by the longword address.	Steal	36.4.1.5.3/36-25	0x1900—byte 0x1940—word 0x1980—lword
Write memory location	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	36.4.1.5.4/36-26	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	36.4.1.5.5/36-28	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	36.4.1.5.6/36-30	0x1C00—byte 0x1C40—word 0x1C80—lword
Resume execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	Halted	36.4.1.5.7/36-31	0x0C00
No operation	NOP	Perform no operation; may be used as a null command.	Parallel	36.4.1.5.8/36-32	0x0000
Output the current PC	SYNC_PC	Capture the current PC and display it on the PST/DDATA outputs.	Parallel	36.4.1.5.9/36-32	0x0001
Read control register	RCREG	Read the system control register.	Halted	36.4.1.5.10/36-33	0x2980
Write control register	WCREG	Write the operand data to the system control register.	Halted	36.4.1.5.13/36-35	0x2880
Read debug module register	RDMREG	Read the debug module register.	Parallel	36.4.1.5.14/36-36	0x2D {0x4[2] DRc[4:0]}
Write debug module register	WDMREG	Write the operand data to the debug module register.	Parallel	36.4.1.5.15/36-37	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 36-20.

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 36.4.1.2, "BDM Serial Interface," describes the receive packet format.

36.4.1.5.1   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 36-17. RAREG/RDREG Command Format
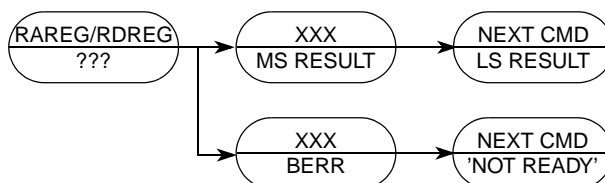
Command Sequence:

Figure 36-18. 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.

36.4.1.5.2   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]															
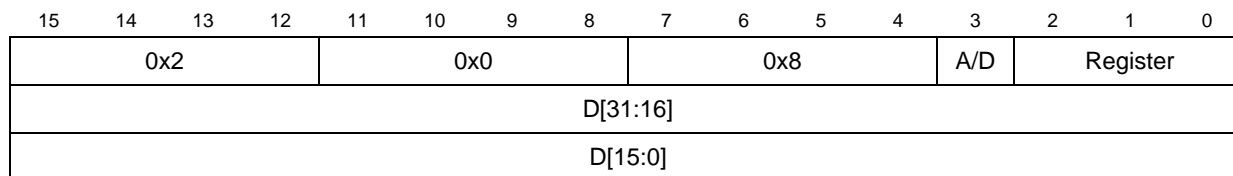
Figure 36-19. WAREG/WDREG Command Format
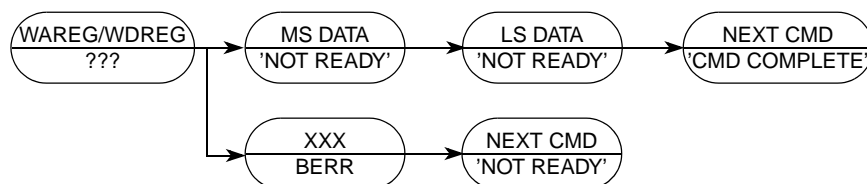
Command Sequence:

Figure 36-20. 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.

36.4.1.5.3   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 36-21. READ Command/Result Formats
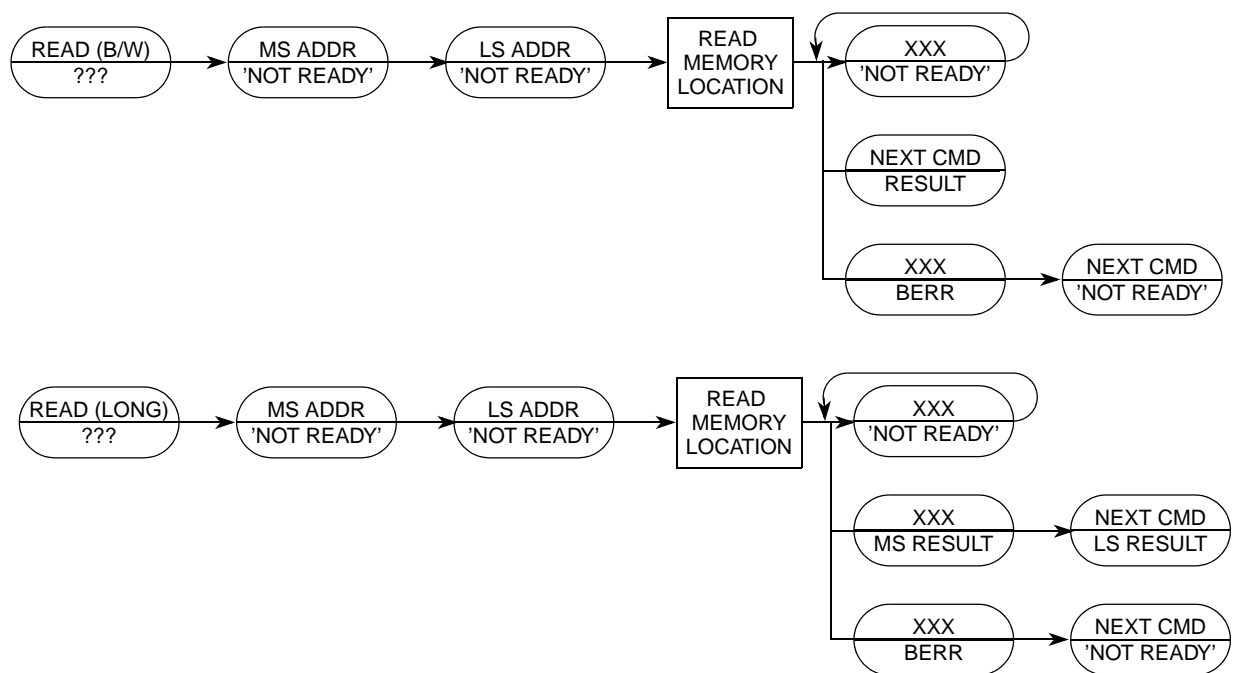
Command Sequence:

Figure 36-22. 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.

36.4.1.5.4   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:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1				0x8				0x0				0x0			
	A[31:16]															
	A[15:0]															
	X	X	X	X	X	X	X	X	D[7:0]							
Word	0x1				0x8				0x4				0x0			
	A[31:16]															
	A[15:0]															
	D[15:0]															
Longword	0x1				0x8				0x8				0x0			
	A[31:16]															
	A[15:0]															
	D[31:16]															
	D[15:0]															

Figure 36-23. WRITE Command Format

Command Sequence:
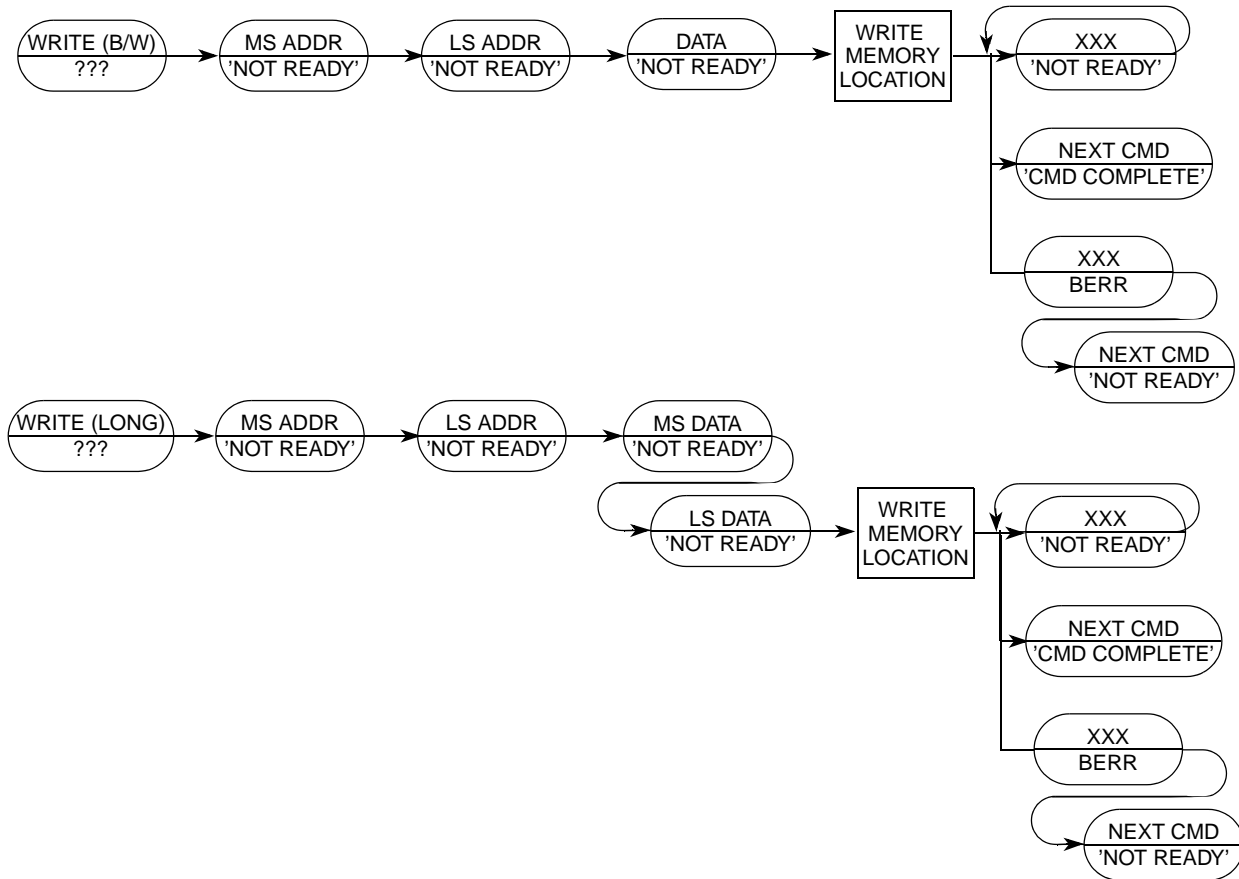
Figure 36-24. WRITE Command Sequence

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.

36.4.1.5.5   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 36-25. DUMP Command/Result Formats
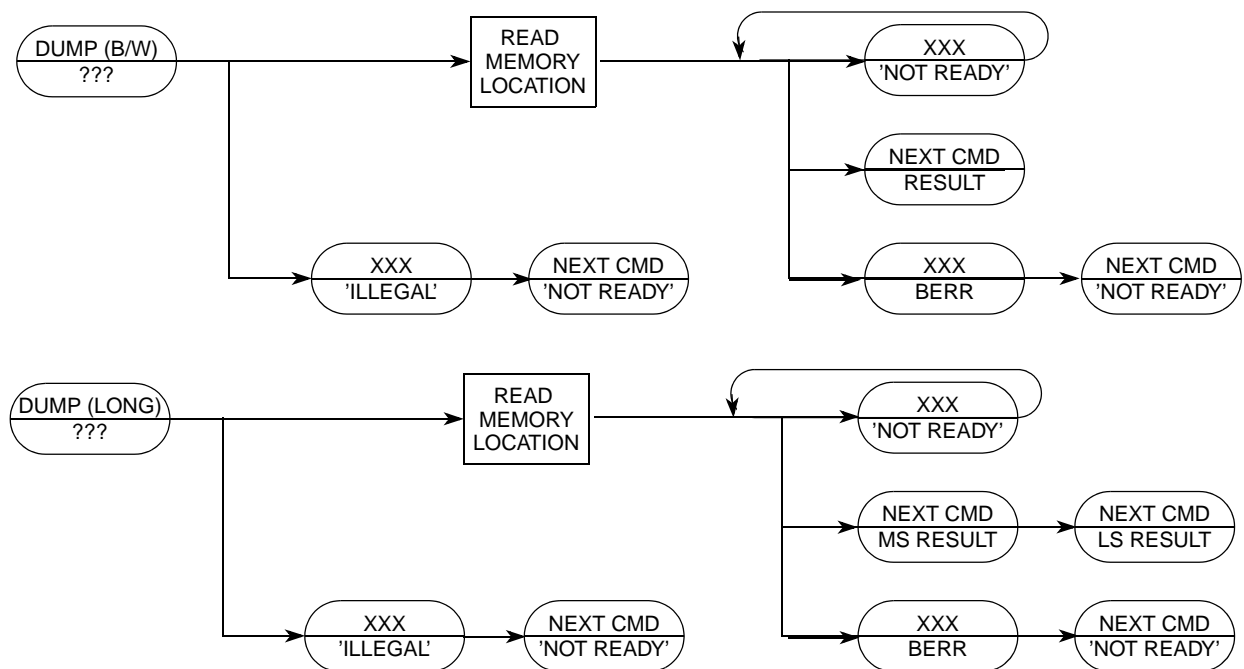
Command Sequence:

Figure 36-26. 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.

36.4.1.5.6 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 36-27. FILL **Command Format**

MCF5329 Reference Manual, Rev 3

36-30   Freescale Semiconductor
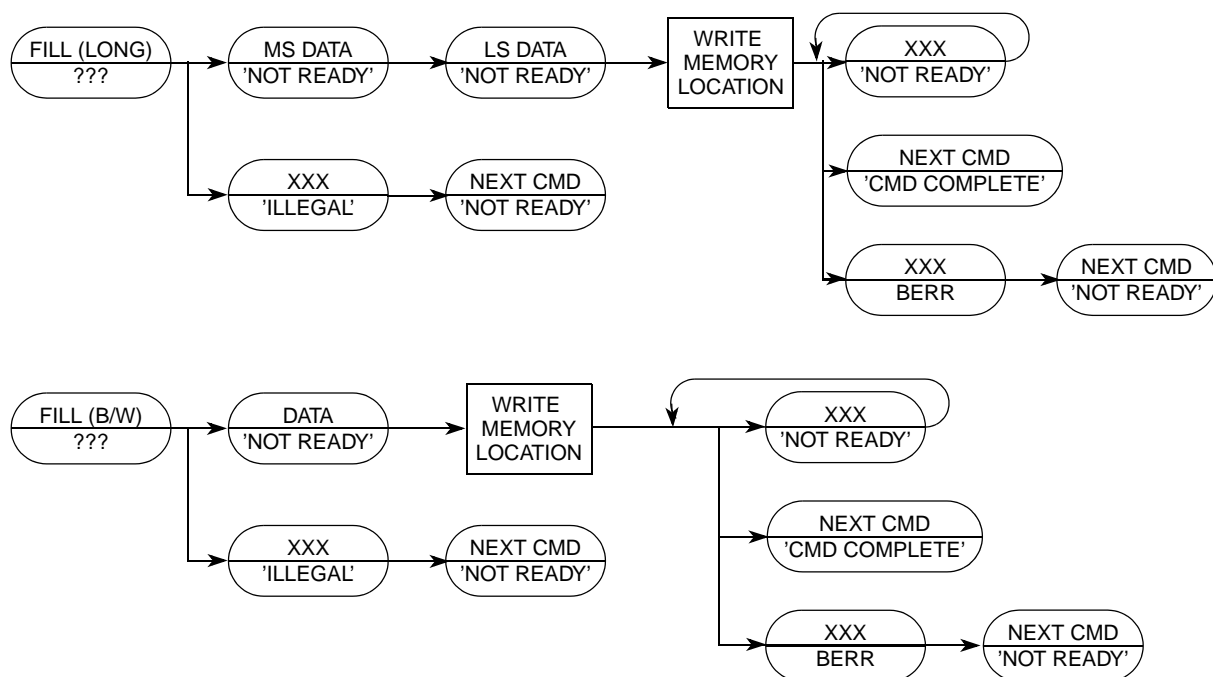
Command Sequence:

Figure 36-28. 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.

36.4.1.5.7   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		
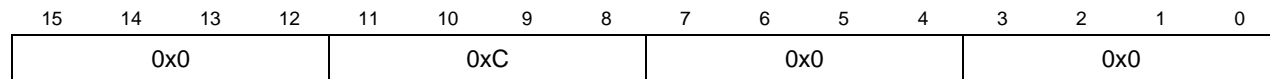
Figure 36-29. GO Command Format

Command Sequence:
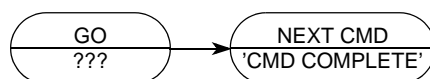
Figure 36-30. GO Command Sequence

Operand Data: None

Result Data: The command-complete response (0xFFFF) is returned during the next shift
                       operation.

36.4.1.5.8 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			
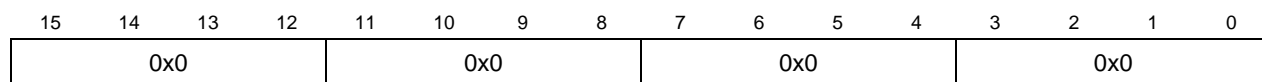
Figure 36-31. NOP Command Format

Command Sequence:
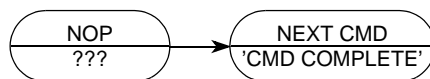
Figure 36-32. NOP Command Sequence

Operand Data: None

Result Data: The command-complete response, 0xFFFF (with S cleared), is returned during the
                       next shift operation.

36.4.1.5.9 Synchronize PC to the PST/DDATA Lines (SYNC_PC)

The SYNC_PC command captures the current PC and displays it on the PST/DDATA 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 PST and DDATA
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.

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		
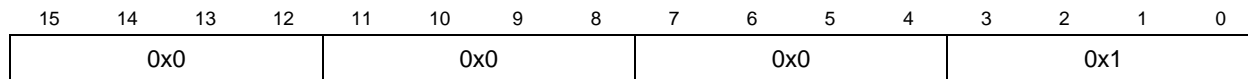
Figure 36-33. SYNC_PC Command Format

Command Sequence:
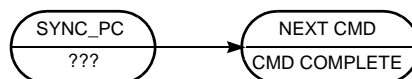
Figure 36-34. SYNC_PC Command Sequence

Operand Data: None

Result Data: Command complete status (0xFFFF) is returned when the register write is
                     complete.

36.4.1.5.10   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]								
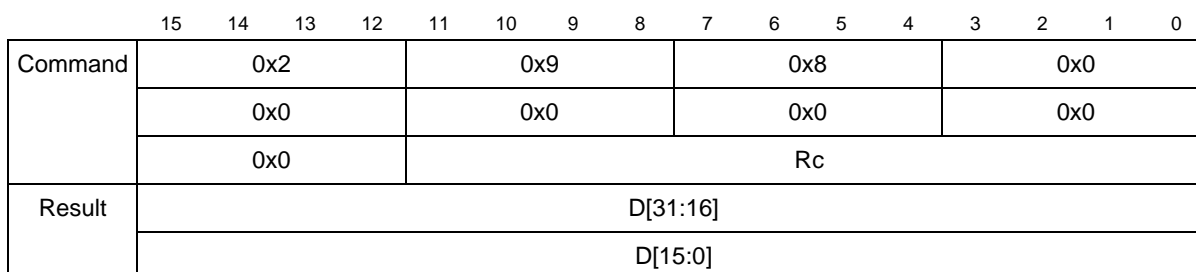
Figure 36-35. RCREG Command/Result Formats
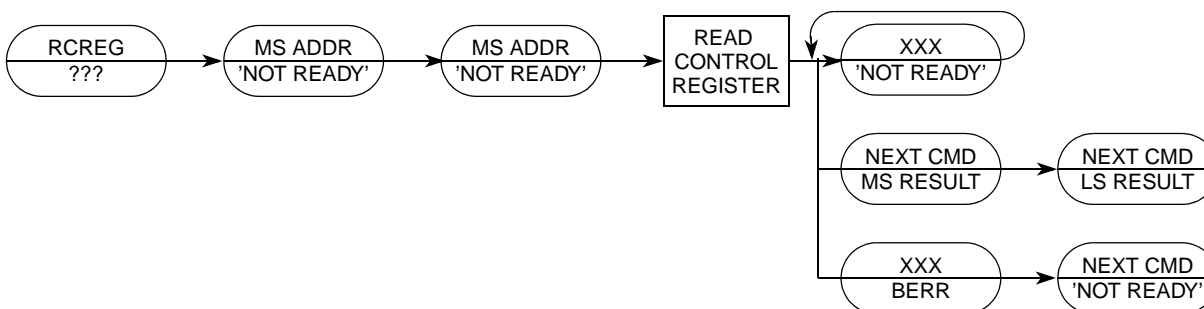
Command Sequence:

Figure 36-36. 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 36-21.

Table 36-21. Control Register Map

Rc	Register Definition
0x002	Cache Control Register (CACR)
0x004	Access Control Register (ACR0)
0x005	Access Control Register (ACR1)
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)
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.

36.4.1.5.11   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.

36.4.1.5.12   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 4.3.1.2, "Saving and Restoring the EMAC Programming Model."

36.4.1.5.13   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 36-37. WCREG Command/Result Formats

MCF5329 Reference Manual, Rev 3
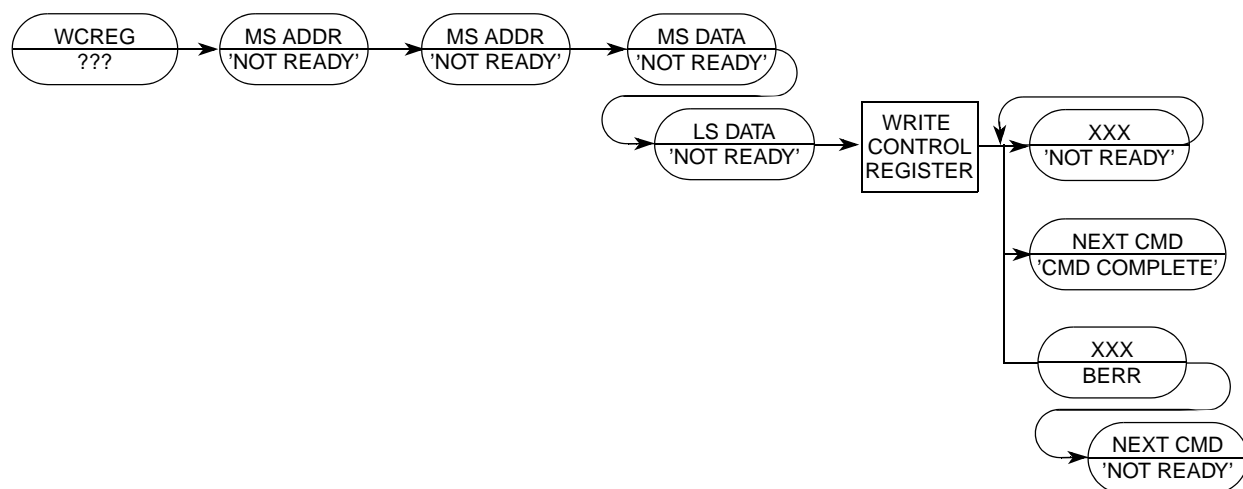
Command Sequence:

Figure 36-38. 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.

36.4.1.5.14   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.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command		0x2				0xD				100				DRc		
Result								D[31:16]								
								D[15:0]								

Figure 36-39. RDMREG Command/Result Formats

Table 36-22 shows the definition of DRc encoding.

Table 36-22. Definition of DRc Encoding—Read

DRc[4:0]	Debug Register Definition	Mnemonic
0x00	Configuration/Status	CSR

Command Sequence:
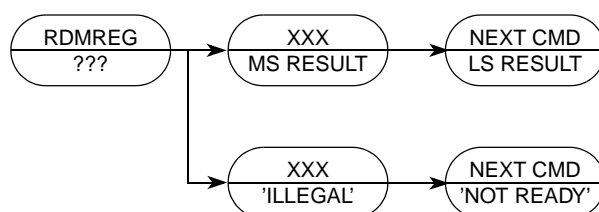
Figure 36-40. RDMREG Command Sequence

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.

36.4.1.5.15 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 36-41. WDMREG BDM Command Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0xC				100			DRc				
D[31:16]															
D[15:0]															
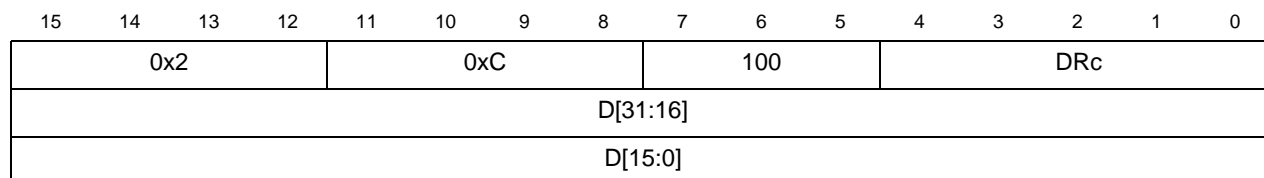
Table 36-3 shows the definition of the DRc write encoding.

Command Sequence:
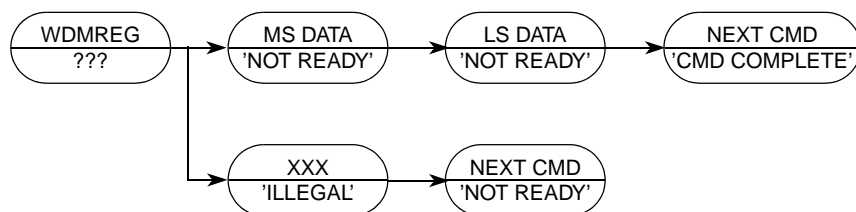
Figure 36-42. WDMREG Command Sequence

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.

36.4.2 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.

36.4.2.1 Theory of Operation

Breakpoint hardware can be configured through TDR[TCR] to respond to triggers by displaying DDATA, initiating a processor halt, or generating a debug interrupt. As shown in Table 36-23, when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the DDATA output port when it is not displaying captured processor status, operands, or branch addresses.

Table 36-23. DDATA[3:0]/CSR[BSTAT] Breakpoint Response

DDATA[3:0][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 TDR 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, 12, from the vector table.Refer to the *ColdFire Programmer's Reference Manual*. for more information.

Execution continues at the instruction address in the vector corresponding to the debug interrupt. All interrupts are ignored while the processor is in emulator mode. 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.

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 revision B/B+, 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.

36.4.2.2   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 36.4.1.1, "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:

- All interrupts are ignored, including level-7 interrupts.
- 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).

36.4.3 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 should be disabled while breakpoint registers are loaded, after which TDR 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).

NOTE
The debug module requires the use of the internal bus to perform BDM commands. For this processor core, if the processor is executing a tight loop contained within a single aligned longword, the processor may never grant the internal bus to the debug module, for example:

```
        align4
label1: nop
        bra.b label1
or
        align4
label2: bra.w label2
```
The processor grants the internal bus if these loops are forced across two longwords.

36.4.4 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 port is partitioned into two 4-bit nibbles: one nibble allows the processor to transmit processor status, (PST), and the other allows operand data to

be displayed (debug data, DDATA). The processor status may not be related to the current bus transfer, due to the decoupling FIFOs.

External development systems can use PST 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). DDATA outputs can display the target address of such instructions in sequential nibble increments across multiple processor clock cycles, as described in Section 36.4.4.1, "Begin Execution of Taken Branch (PST = 0x5)". Two 32-bit storage elements form a FIFO buffer connecting the processor's high-speed local bus to the external development system through PST[3:0] and DDATA[3:0]. The buffer captures branch target addresses and certain data values for eventual display on the DDATA port, one nibble at a time starting with the least significant bit (lsb).

Execution speed is affected only when both storage elements contain valid data to be dumped to the DDATA port. The core stalls until one FIFO entry is available.

Table 36-24 shows the encoding of these signals.

Table 36-24. 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 PST 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	Used by the debug translate module to indicate two consecutive PST = 1 values.
0x3	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
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 DDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled on the PST port, followed by the appropriate marker, and then the data transfer on the DDATA port. Transfer length depends on the WDDATA operand size.
0x5	Begin execution of taken branch or SYNC_PC command issued. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. See Section 36.4.4.1, "Begin Execution of Taken Branch (PST = 0x5)". Also indicates that the SYNC_PC command has been issued.
0x6	Used by the debug translate module to indicate a PST = 5 value followed by a PST = 1 value.
0x7	Begin execution of return from exception (RTE) instruction.
0x8– 0xB	Indicates the number of bytes to be displayed on the DDATA port on subsequent clock cycles. The value is driven onto the PST port one PSTCLK cycle before the data is displayed on DDATA. 0x8 Begin 1-byte transfer on DDATA. 0x9 Begin 2-byte transfer on DDATA. 0xA Begin 3-byte transfer on DDATA. 0xB Begin 4-byte transfer on DDATA.

Table 36-24. Processor Status Encoding (continued)

PST[3:0]	Definition
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, PST 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, PST outputs are driven with 0xD until exception processing completes.
0xE	Processor is stopped. Appears in multiple-cycle format when the processor executes a STOP instruction. The ColdFire processor remains stopped until an interrupt occurs, thus PST outputs display 0xE until the stopped mode is exited.
0xF	Processor is halted. Because this encoding defines a multiple-cycle mode, the PST outputs display 0xF until the processor is restarted or reset. See Section 36.4.1.1, "CPU Halt".

36.4.4.1 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 DDATA 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 DDATA 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 PST (0x5) to identify that a taken branch is executed.
2. Using the PST pins, optionally signal the target address to be displayed sequentially on the DDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.
3. The new target address is optionally available on subsequent cycles using the DDATA port. The number of bytes of displayed on this port is configurable (2, 3, or 4 bytes, where the DDATA encoding is 0x9, 0xA, and 0xB, respectively).

Another example of a variant branch instruction would be a JMP (A0) instruction. Figure 36-43 shows the PST and DDATA outputs that indicate a JMP (A0) execution, assuming the CSR was programmed to display the lower 2 bytes of an address.
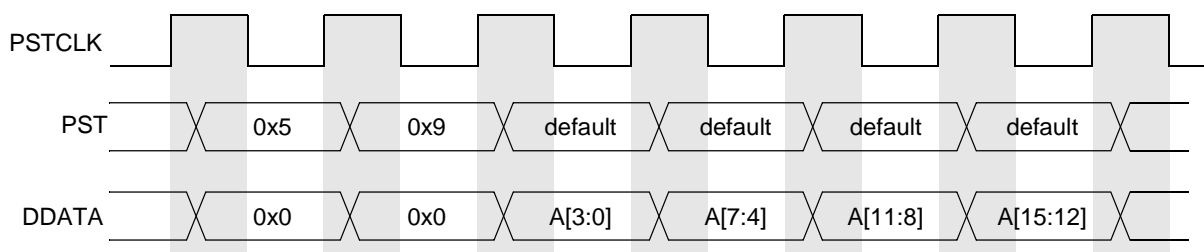
Figure 36-43. Example JMP Instruction Output on PST/DDATA

PST of 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Therefore, the subsequent 4 nibbles of DDATA display the lower two bytes of address register A0 in least-to-most-significant nibble order. The PST output after the JMP instruction completes depends on the target instruction. The PST can continue with the next instruction before the address has completely displayed on DDATA because of the DDATA FIFO. If the FIFO is full and the next instruction has captured values to display on DDATA, the pipeline stalls (PST = 0x0) until space is available in the FIFO.

36.4.5 Debug Translate Block

With a maximum core operating speed of 240 MHz, the ColdFire debug interface on PST/DDATA must run slower to support emulator technology. This is accomplished with the debug translate block, which effectively time shifts the PST/DDATA output stream into a logically equivalent 1/2 speed PST/DDATA output.

The specific rules of PST/DDATA compression are as follows:

- Two consecutive PST = 1 values can be compressed to a single PST value, where PST = 2.
- A value of PST = 5 followed by a value of PST = 1 can be compressed to a single PST = 6 value.
- PST encodings that are asserted for multiple clock cycles (PST= 0xC, 0xD, 0xE, 0xF) are compressed as required. For example, two PST = 0xC values can be compressed into a single PST = 0xC.
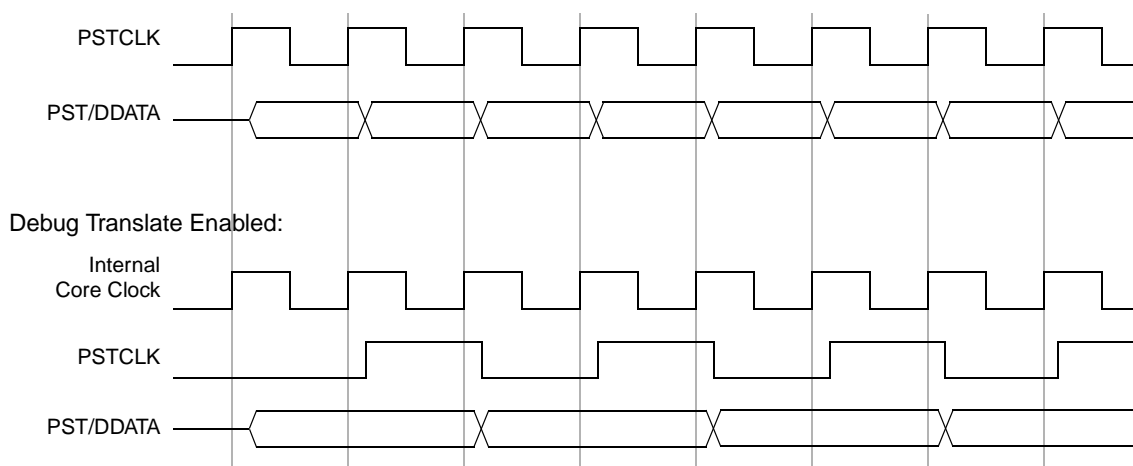
Figure 36-44. Debug Translate Timing

36.4.6 Processor Status, Debug Data Definition

This section specifies the ColdFire processor and debug module's generation of the processor status (PST) and debug data (DDATA) output on an instruction basis. In general, the PST/DDATA output for an instruction is defined as follows:

PST = 0x1, {PST = [0x89B], DDATA = 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 DDATA 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 DDATA output {2, 3, or 4 bytes} using a PST value of {0x9, 0xA, or 0xB}.

36.4.6.1 User Instruction Set

Table 36-25 shows the PST/DDATA 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. The DD nomenclature refers to the DDATA outputs.

Table 36-25. PST/DDATA Specification for User-Mode Instructions

Instruction	Operand Syntax	PST/DDATA
add.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
add.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
adda.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source operand}
addi.l	#<data>,Dx	PST = 0x1
addq.l	#<data>,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
addx.l	Dy,Dx	PST = 0x1
and.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
and.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
andi.l	#<data>,Dx	PST = 0x1
asl.l	{Dy,#<data>},Dx	PST = 0x1
asr.l	{Dy,#<data>},Dx	PST = 0x1
bcc.{b,w}		if taken, then PST = 0x5, else PST = 0x1
bchg.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bchg.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bitrev.l	Dx	PST = 0x1

Table 36-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
bra.{b,w}		PST = 0x5
bset.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bset.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bsr.{b,w}		PST = 0x5, {PST = 0xB, DD = destination operand}
btst.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
btst.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
byterev.l	Dx	PST = 0x1
clr.b	<ea>x	PST = 0x1, {PST = 0x8, DD = destination operand}
clr.l	<ea>x	PST = 0x1, {PST = 0xB, DD = destination operand}
clr.w	<ea>x	PST = 0x1, {PST = 0x9, DD = destination operand}
cmp.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
cmpa.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source operand}
cmpi.l	#<data>,Dx	PST = 0x1
divs.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divs.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
divu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
eor.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
eori.l	#<data>,Dx	PST = 0x1
ext.l	Dx	PST = 0x1
ext.w	Dx	PST = 0x1
extb.l	Dx	PST = 0x1
illegal		PST = 0x1[1]
jmp	<ea>y	PST = 0x5, {PST = [0x9AB], DD = target address}[2]
jsr	<ea>y	PST = 0x5, {PST = [0x9AB], DD = target address}, {PST = 0xB, DD = destination operand}[2]
lea.l	<ea>y,Ax	PST = 0x1
link.w	Ay,#<displacement>	PST = 0x1, {PST = 0xB, DD = destination operand}
lsl.l	{Dy,#<data>},Dx	PST = 0x1
lsr.l	{Dy,#<data>},Dx	PST = 0x1
move.b	<ea>y,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x1, {PST = 0x9, DD = source}, {PST = 0x9, DD = destination}

Table 36-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
move.w	CCR,Dx	PST = 0x1
move.w	{Dy,#<data>},CCR	PST = 0x1
movea.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source}
movea.w	<ea>y,Ax	PST = 0x1, {PST = 0x9, DD = source}
movem.l	#list,<ea>x	PST = 0x1, {PST = 0xB, DD = destination},... [3]
movem.l	<ea>y,#list	PST = 0x1, {PST = 0xB, DD = source},... [3]
moveq.l	#<data>,Dx	PST = 0x1
muls.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
neg.l	Dx	PST = 0x1
negx.l	Dx	PST = 0x1
nop		PST = 0x1
not.l	Dx	PST = 0x1
or.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
or.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
ori.l	#<data>,Dx	PST = 0x1
pea.l	<ea>y	PST = 0x1, {PST = 0xB, DD = destination operand}
pulse		PST = 0x4
rems.l	<ea>y,Dw:Dx	PST = 0x1, {PST = 0xB, DD = source operand}
remu.l	<ea>y,Dw:Dx	PST = 0x1, {PST = 0xB, DD = source operand}
rts		PST = 0x1, {PST = 0xB, DD = source operand}, PST = 0x5, {PST = [0x9AB], DD = target address}
scc.b	Dx	PST = 0x1
sub.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
sub.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
suba.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = source operand}
subi.l	#<data>,Dx	PST = 0x1
subq.l	#<data>,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
subx.l	Dy,Dx	PST = 0x1
swap.w	Dx	PST = 0x1
tpf		PST = 0x1

Table 36-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
tpf.l	#\<data\>	PST = 0x1
tpf.w	#\<data\>	PST = 0x1
trap	#\<data\>	PST = 0x1[1]
tst.b	\<ea\>x	PST = 0x1, {PST = 0x8, DD = source operand}
tst.l	\<ea\>y	PST = 0x1, {PST = 0xB, DD = source operand}
tst.w	\<ea\>y	PST = 0x1, {PST = 0x9, DD = source operand}
unlk	Ax	PST = 0x1, {PST = 0xB, DD = destination operand}
wddata.b	\<ea\>y	PST = 0x4, {PST = 0x8, DD = source operand
wddata.l	\<ea\>y	PST = 0x4, {PST = 0xB, DD = source operand
wddata.w	\<ea\>y	PST = 0x4, {PST = 0x9, DD = source operand

[1] During normal exception processing, the PST output is driven to a 0xC 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:
    PST = 0xC,
    {PST = 0xB,DD = destination},          // stack frame
    {PST = 0xB,DD = destination},          // stack frame
    {PST = 0xB,DD = source},               // vector read
    PST = 0x5,{PST = [0x9AB],DD = target}// handler PC
```

The PST/DDATA specification for the reset exception is shown below:

```
Exception Processing:
    PST = 0xC,
    PST = 0x5,{PST = [0x9AB],DD = 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 PST = 0xC value is driven at all times, unless the PST 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 36-26 shows the PST/DDATA specification for multiply-accumulate instructions.

Table 36-26. PST/DDATA Values for User-Mode Multiply-Accumulate Instructions

Instruction	Operand Syntax	PST/DDATA
mac.l	Ry,Rx,ACCx	PST = 0x1
mac.l	Ry,Rx,\<ea\>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}

MCF5329 Reference Manual, Rev 3

Table 36-26. PST/DDATA Values for User-Mode Multiply-Accumulate Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
mac.w	Ry,Rx,ACCx	PST = 0x1
mac.w	Ry,Rx,ea,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}
move.l	{Ry,#<data>},ACCx	PST = 0x1
move.l	{Ry,#<data>},MACSR	PST = 0x1
move.l	{Ry,#<data>},MASK	PST = 0x1
move.l	{Ry,#<data>},ACCext01	PST = 0x1
move.l	{Ry,#<data>},ACCext23	PST = 0x1
move.l	ACCext01,Rx	PST = 0x1
move.l	ACCext23,Rx	PST = 0x1
move.l	ACCy,ACCx	PST = 0x1
move.l	ACCy,Rx	PST = 0x1
move.l	MACSR,CCR	PST = 0x1
move.l	MACSR,Rx	PST = 0x1
move.l	MASK,Rx	PST = 0x1
msac.l	Ry,Rx,ACCx	PST = 0x1
msac.l	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}
msac.w	Ry,Rx,ACCx	PST = 0x1
msac.w	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = source operand}

36.4.6.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. The PST/DDATA specification for these opcodes is shown in Table 36-27.

Table 36-27. PST/DDATA Specification for Supervisor-Mode Instructions

Instruction	Operand Syntax	PST/DDATA
cpushl	(Ax)	PST = 0x1
halt		PST = 0x1, PST = 0xF
move.l	Ay,USP	PST = 0x1
move.l	USP,Ax	PST = 0x1
move.w	SR,Dx	PST = 0x1
move.w	{Dy,#<data>},SR	PST = 0x1, {PST = 0x3}
movec.l	Ry,Rc	PST = 0x1

Table 36-27. PST/DDATA Specification for Supervisor-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
rte		PST = 0x7, {PST = 0xB, DD = source operand}, {PST = 0x3},{ PST = 0xB, DD =source operand}, PST = 0x5, {[PST = 0x9AB], DD = target address}
stop	#<data>	PST = 0x1, PST = 0xE
wdebug.l	<ea>y	PST = 0x1, {PST = 0xB, DD = source, PST = 0xB, DD = source}

The move-to-SR and RTE instructions include an optional PST = 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 (PST = 0xE) and the halted state (PST = 0xFF) display this status throughout the entire time the ColdFire processor is in the given mode.

36.4.7   Freescale-Recommended BDM Pinout

The ColdFire BDM connector is a 26-pin Berg connector arranged 2 x 13 as shown below.
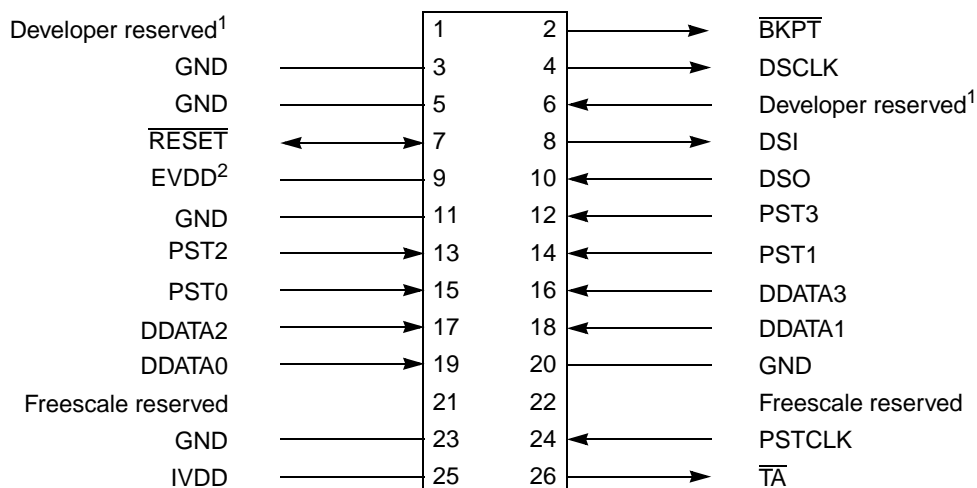
[1] Pins reserved for BDM developer use.
[2] Supplied by target

Figure 36-45. Recommended BDM Connector

Chapter 37
IEEE 1149.1 Test Access Port (JTAG)

37.1 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, $\overline{TRST}$.

37.1.1 Block Diagram

Figure 37-1 shows the block diagram of the JTAG module.
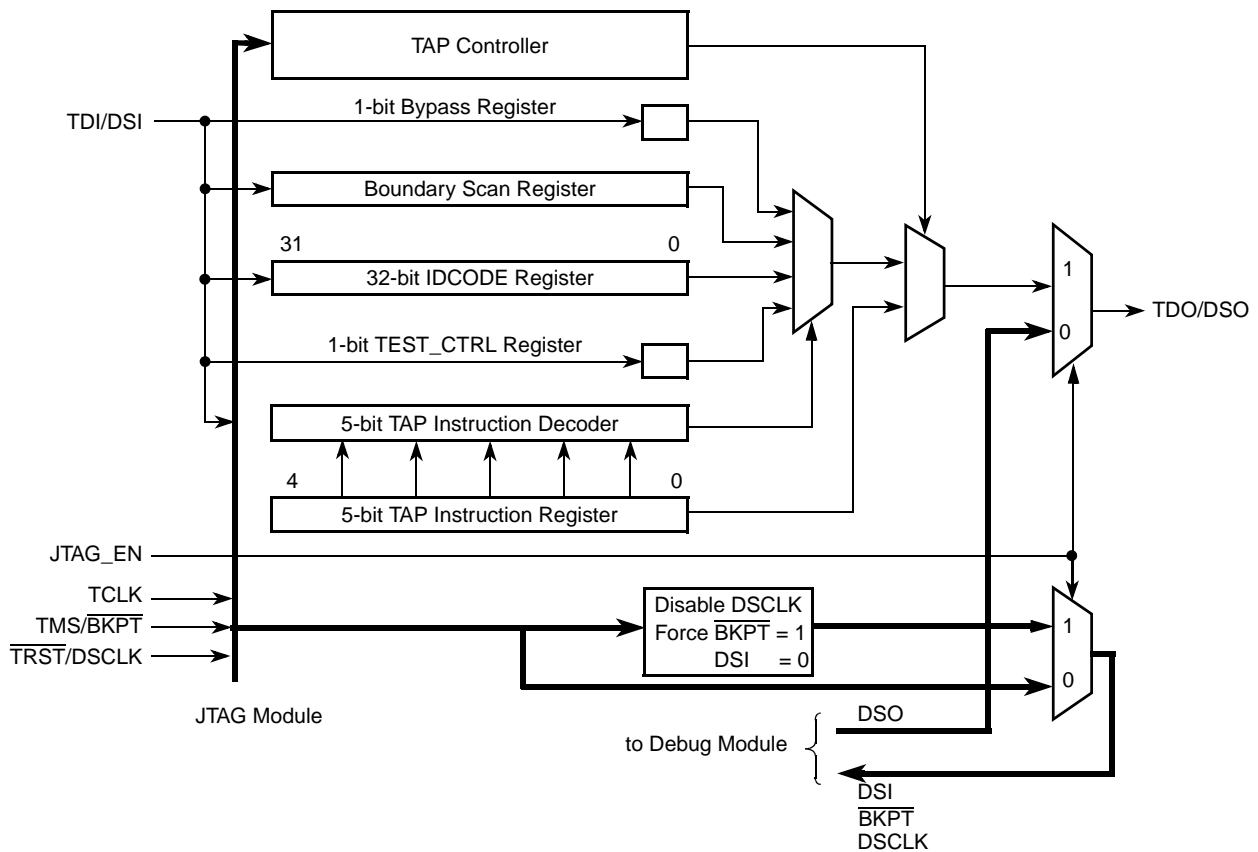
Figure 37-1. JTAG Block Diagram

37.1.2 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

37.1.3 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 36.4.1, "Background Debug Mode (BDM)"; (JTAG_EN = 0).

37.2 External Signal Description

The JTAG module has five input and one output external signals, as described in Table 37-1.

Table 37-1. 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	—

37.2.1 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 37-2 summarizes the pin function selected depending on JTAG_EN logic state.

Table 37-2. 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 $\overline{\text{TDO}}$ $\overline{\text{TRST}}$	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 37-3, to disable the corresponding module.

Table 37-3. 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.

37.2.2 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.

37.2.3 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.

37.2.4 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.

37.2.5 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.

37.2.6 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.

37.3 Memory Map/Register Definition

The JTAG module registers are not memory mapped and are only accessible through the TDO/DSO pin.

37.3.1 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 37.4.3, "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 37-2. 5-Bit Instruction Register (IR)

37.3.2 IDCODE Register

The IDCODE is a read-only register; its value is chip dependent. For more information, see Section 37.4.3.1, "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
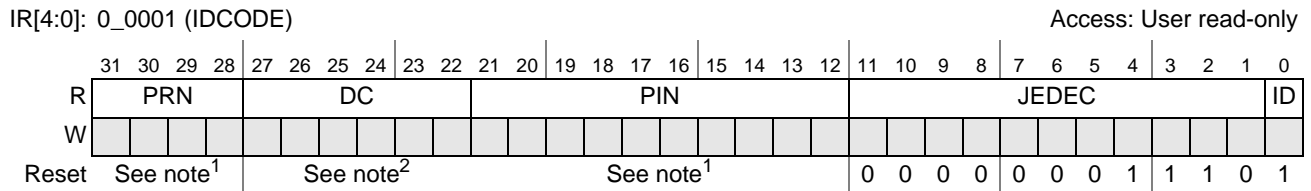
[1] The reset values for PRN and PIN are device-dependent.
[2] Varies, depending on design center location.

Figure 37-3. IDCODE Register

Table 37-4. 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. 0x061 MCF5327 0x059 MCF5328 0x058 MCF53281 0x054 MCF5329
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.

37.3.3 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.

37.3.4 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 37-4. 1-Bit TEST_CTRL Register

37.3.5 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.

37.4 Functional Description

37.4.1 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.

37.4.2 TAP Controller

The TAP controller is a state machine that changes state based on the sequence of logical values on the TMS pin. Figure 37-5 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 $\overline{\text{TRST}}$ signal asynchronously resets the TAP controller to the test-logic-reset state. As Figure 37-5 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.
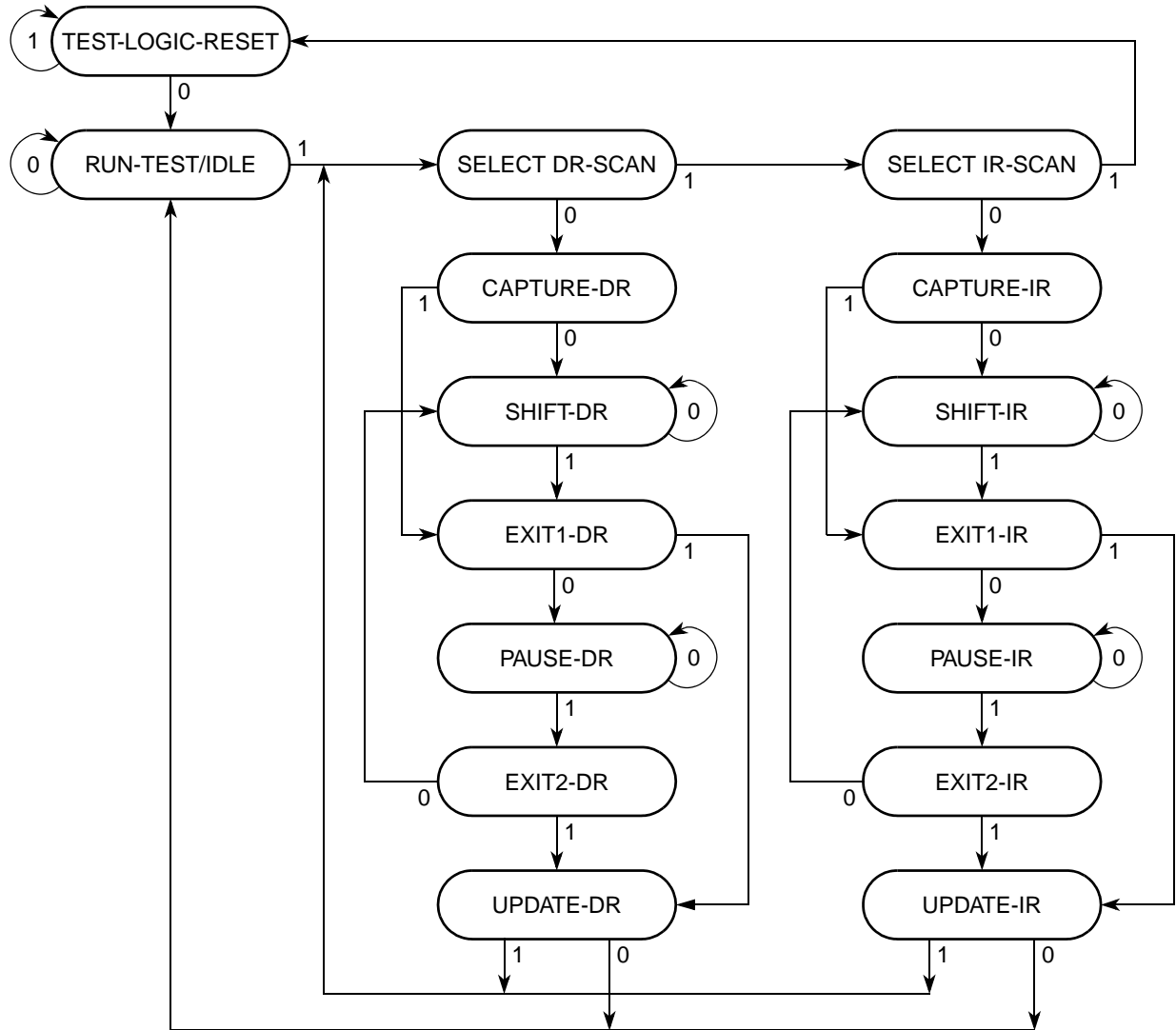
Figure 37-5. TAP Controller State Machine Flow

37.4.3 JTAG Instructions

Table 37-5 describes public and private instructions.

Table 37-5. 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 37-5. 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.

37.4.3.1 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.

37.4.3.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- SAMPLE - See Section 37.4.3.3, "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.

37.4.3.3 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.

37.4.3.4 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.

37.4.3.5 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.

37.4.3.6 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.

37.4.3.7 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.

37.4.3.8 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.

37.5 Initialization/Application Information

37.5.1 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 $\overline{TRST}$ 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.

37.5.2 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 $\overline{TRST}$ be pulled up. $\overline{TRST}$ could be connected to ground. However, because there is a pull-up on $\overline{TRST}$, 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 $\overline{TRST}$.

Appendix A
Register Memory Map Quick Reference

A.1   Register Memory Map

Table A-1 illustrates the overall device memory map. Table A-2 and Table A-3 list the base address for each peripheral within the two peripheral controllers space. Each module is then detailed in Table A-6 through Table A-34. Table A-4 and Table A-5 summarize the ColdFire core and debug registers, which are not accessible through the memory map but are included here for completeness.

Table A-1. Device Memory Map Overview

Address Range	Module	Size
0x0000_0000–0x3FFF_FFFF	FlexBus	1024 MB
0x4000_0000–0x7FFF_FFFF	SDRAM Controller	1024 MB
0x8000_0000–0x8FFF_FFFF	Internal SRAM Backdoor	256 MB
0xC000_0000–0xDFFF_FFFF	FlexBus	512 MB
0xE000_0000–0xEFFF_FFFF[1]	On-chip Peripheral Controller 1	256 MB
0xF000_0000–0xFFFF_FFFF[1]	On-chip Peripheral Controller 0	256 MB

[1]   See the various tables below or the peripheral chapters for their memory maps. Any unused space by these peripherals within this memory range is reserved and should 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) 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, e.g., ADDR[31]=0 identifies the cacheable space.

Table A-2. Peripheral Controller 0 Base Addresses

Base Address	Slot Number	Peripheral	Module Memory Map
0xFC00_0000	0	SCM (MPR1, PACRs, & BMT1)	Table A-6
0xFC00_4000	1	Cross-bar switch	Table A-7
0xFC00_8000	2	FlexBus	Table A-8
0xFC02_0000	8	FlexCAN	Table A-9
0xFC03_0000	12	FEC	Table A-10
0xFC04_0000	16	SCM (CWT & Core Fault Registers)	Table A-6
0xFC04_4000	17	eDMA Controller	Table A-11
0xFC04_8000	18	Interrupt Controller 0	
0xFC04_C000	19	Interrupt Controller 1	Table A-12
0xFC05_4000	21	Interrupt Controller IACK	
0xFC05_8000	22	I^{2}C	Table A-13
0xFC05_C000	23	QSPI	Table A-14
0xFC06_0000	24	UART0	
0xFC06_4000	25	UART1	Table A-15
0xFC06_8000	26	UART2	
0xFC07_0000	28	DMA Timer 0	
0xFC07_4000	29	DMA Timer 1	Table A-16
0xFC07_8000	30	DMA Timer 2	
0xFC07_C000	31	DMA Timer 3	
0xFC08_0000	32	PIT 0	
0xFC08_4000	33	PIT 1	Table A-17
0xFC08_8000	34	PIT 2	
0xFC08_C000	35	PIT 3	
0xFC09_0000	36	PWM	Table A-18
0xFC09_4000	37	Edge Port	Table A-19
0xFC09_8000	38	On-chip Watchdog Timer	Table A-20
0xFC0A_0000	40	CCM, Reset Controller, Power Management	Table A-21, Table A-22, Table A-23
0xFC0A_4000	41	GPIO Module	Table A-24
0xFC0A_8000	42	Real Time Clock	Table A-25
0xFC0A_C000	43	LCD Controller	Table A-26
0xFC0B_0000	44	USB On-the-Go	Table A-27

Table A-2. Peripheral Controller 0 Base Addresses (continued)

Base Address	Slot Number	Peripheral	Module Memory Map
0xFC0B_4000	45	USB Host	Table A-28
0xFC0B_8000	46	SDRAM Controller	Table A-29
0xFC0B_C000	47	SSI	Table A-30
0xFC0C_0000	48	PLL	Table A-31

Table A-3. Peripheral Controller 1 Base Addresses

Base Address	Slot Number	Peripheral	Module Memory Map
0xEC00_0000	0	SCM (MPR0 & BMT0)	Table A-6
0xEC08_0000	56	MDHA	Table A-32
0xEC08_4000	57	SKHA	Table A-33
0xEC08_8000	58	RNG	Table A-34

Table A-4. 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	0xCF30_60	No	3.2.1/3-6
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x0000_0670	No	3.2.1/3-6
Load: 0x082–7 Store: 0x182–7	Data Register 2–7 (D2–D7)	32	R/W	Undefined	No	3.2.1/3-6
Load: 0x088–8E Store: 0x188–8E	Address Register 0–6 (A0–A6)	32	R/W	Undefined	No	3.2.2/3-6
Load: 0x08F Store: 0x18F	Supervisor/User A7 Stack Pointer (A7)	32	R/W	Undefined	No	3.2.3/3-6
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	No	4.2.1/4-3
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	No	4.2.2/4-5
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	No	4.2.3/4-6
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	No	4.2.4/4-7
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	No	4.2.4/4-7
0x80E	Condition Code Register (CCR)	8	R/W	Undefined	No	3.2.4/3-7

Table A-4. ColdFire Core Programming Model (continued)

BDM[1]	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
0x80F	Program Counter (PC)	32	R/W	Contents of location 0x0000_0004	No	3.2.5/3-8
Supervisor Access Only Registers						
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	3.2.6/3-8
0x004–5	Access Control Register 0–1 (ACR0–1)	32	R/W	See Section	Yes	3.2.7/3-9
0x800	User/Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x0000_0000	No	3.2.3/3-6
0x801	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes	3.2.8/3-9
0x80E	Status Register (SR)	16	R/W	0x27--	No	3.2.9/3-9
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	3.2.10/3-10

[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 36, "Debug Module".

Table A-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	0x0098_0000	36.3.2/36-5
0x05	BDM address attribute register (BAAR)	32[1]	W	0x05	36.3.3/36-8
0x06	Address attribute trigger register (AATR)	32[1]	W	0x0005	36.3.4/36-9
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	36.3.5/36-10
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined	36.3.6/36-13
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined	36.3.6/36-13
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined	36.3.7/36-15
0x0D	Address breakpoint low register (ABLR)	32	W	Undefined	36.3.7/36-15
0x0E	Data breakpoint register (DBR)	32	W	Undefined	36.3.8/36-16
0x0F	Data breakpoint mask register (DBMR)	32	W	Undefined	36.3.8/36-16
0x18	PC breakpoint register 1 (PBR1)	32	W	See Section	36.3.6/36-13
0x1A	PC breakpoint register 2 (PBR2)	32	W	See Section	36.3.6/36-13
0x1B	PC breakpoint register 3 (PBR3)	32	W	See Section	36.3.6/36-13

[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. CSR is write-only from the programming model. It can be read or written through the BDM port using the RDMREG and WDMREG commands.

Table A-6. SCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC00_0000[1]	Master Privilege Register 1 (MPR1)	32	R/W	0x7000_0007	11.2.1/11-3
0xEC00_0054[1]	Bus Monitor Timeout 1 (BMT1)	32	R/W	0x0000_0008	11.2.4/11-7
0xFC00_0000	Master Privilege Register 0 (MPR0)	32	R/W	0x7777_7777	11.2.1/11-3
0xFC00_0020	Peripheral Access Control Register A (PACRA)	32	R/W	0x5444_4444	11.2.3/11-4
0xFC00_0024	Peripheral Access Control Register B (PACRB)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0028	Peripheral Access Control Register C (PACRC)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_002C	Peripheral Access Control Register D (PACRD)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0040	Peripheral Access Control Register E (PACRE)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0044	Peripheral Access Control Register F (PACRF)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0048	Peripheral Access Control Register G (PACRG)	32	R/W	0x4444_4444	11.2.3/11-4
0xEC00_0040[1]	Peripheral Access Control Register H (PACRH)	32	R/W	0x4444_4444	11.2.3/11-4
0xFC00_0054	Bus Monitor Timeout 0 (BMT0)	32	R/W	0x0000_0008	11.2.4/11-7
0xFC04_0013	Wakeup Control Register (WCR)[2]	8	R/W	0x00	8.2.1/8-2
0xFC04_0016	Core Watchdog Control Register (CWCR)	16	R/W	0x0000	11.2.5/11-8
0xFC04_001B	Core Watchdog Service Register (CWSR)	8	R/W	Undefined	11.2.6/11-9
0xFC04_001F	SCM Interrupt Status Register (SCMISR)	8	R/W	0x00	11.2.7/11-10
0xFC04_0024	Burst Configuration Register (BCR)	32	R/W	0x0000_0000	11.2.8/11-10
0xFC04_0070	Core Fault Address Register (CFADR)	32	R	0x0000_0000	11.2.9/11-11
0xFC04_0075	Core Fault Interrupt Enable Register (CFIER)	8	R/W	0x00	11.2.10/11-12
0xFC04_0076	Core Fault Location Register (CFLOC)	8	R	Undefined	11.2.11/11-12
0xFC04_0077	Core Fault Attributes Register (CFATR)	8	R	Undefined	11.2.12/11-12
0xFC04_007C	Core Fault Data Register (CFDTR)	32	R	Undefined	11.2.13/11-13

[1] Take note of register location.

[2] The WCR register is described in Chapter 8, "Power Management."

Table A-7. XBS Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4100	Priority Register Slave 1 (XBS_PRS1)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4110	Control Register Slave 1 (XBS_CRS1)	32	R/W	0x0000_0000	12.4.2/12-5
0xFC00_4400	Priority Register Slave 4 (XBS_PRS4)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4410	Control Register Slave 4 (XBS_CRS4)	32	R/W	0x0000_0000	12.4.2/12-5
0xFC00_4600	Priority Register Slave 6 (XBS_PRS6)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4610	Control Register Slave 6 (XBS_CRS6)	32	R/W	0x0000_0000	12.4.2/12-5
0xFC00_4700	Priority Register Slave 7 (XBS_PRS7)	32	R/W	0x6543_0210	12.4.1/12-4
0xFC00_4710	Control Register Slave 7 (XBS_CRS7)	32	R/W	0x0000_0000	12.4.2/12-5

Table A-8. FlexBus Chip Select Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_8000 + ($n \times$ 0xC)	Chip-Select Address Register (CSARn) $n = 0 - 5$	32	R/W	0x0000_0000	17.3.1/17-4
0xFC00_8004 + ($n \times$ 0xC)	Chip-Select Mask Register (CSMRn) $n = 0 - 5$	32	R/W	0x0000_0000	17.3.2/17-5
0xFC00_8008 + ($n \times$ 0xC)	Chip-Select Control Register (CSCRn) $n = 0 - 5$	32	R/W	See Section	17.3.3/17-6

Table A-9. FlexCAN Memory Map

Address FlexCAN	Register	Width (bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
Supervisor-only Access Registers							
0xFC02_0000	FlexCAN Module Configuration Register (CANMCR)	32	Y	Y	R/W	0xD890_000F	23.3.1/23-6
Supervisor/User Access Registers							
0xFC02_0004	FlexCAN Control Register (CANCTRL)	32	Y	N	R/W	0x0000_0000	23.3.2/23-8
0xFC02_0008	Free Running Timer (TIMER)	32	Y	Y	R/W	0x0000_0000	23.3.3/23-10
0xFC02_0010	Rx Global Mask (RXGMASK)	32	Y	N	R/W	0x1FFF_FFFF	23.3.4/23-11
0xFC02_0014	Rx Buffer 14 Mask (RX14MASk)	32	Y	N	R/W	0x1FFF_FFFF	23.3.4/23-11
0xFC02_0018	Rx Buffer 15 Mask (RX15MASK)	32	Y	N	R/W	0x1FFF_FFFF	23.3.4/23-11
0xFC02_001C	Error Counter Register (ERRCNT)	32	Y	Y	R/W	0x0000_0000	23.3.5/23-12

Table A-9. FlexCAN Memory Map (continued)

Address FlexCAN	Register	Width (bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
0xFC02_0020	Error and Status Register (ERRSTAT)	32	Y	Y	R/W	0x0000_0000	23.3.6/23-13
0xFC02_0028	Interrupt Mask Register (IMASK)	32	Y	Y	R/W	0x0000_0000	23.3.7/23-15
0xFC02_0030	Interrupt Flag Register (IFLAG)	32	Y	Y	R/W	0x0000_0000	23.3.8/23-16
0xFC02_0080	Message Buffers 0–15 (MB0–15)	2048	N	N	R/W	—	23.3.8/23-16

Table A-10. FEC Register Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_0004	Interrupt Event Register (EIR)	32	R/W	0x0000_0000	19.4.2/19-9
0xFC03_0008	Interrupt Mask Register (EIMR)	32	R/W	0x0000_0000	19.4.3/19-11
0xFC03_0010	Receive Descriptor Active Register (RDAR)	32	R/W	0x0000_0000	19.4.4/19-11
0xFC03_0014	Transmit Descriptor Active Register (TDAR)	32	R/W	0x0000_0000	19.4.5/19-12
0xFC03_0024	Ethernet Control Register (ECR)	32	R/W	0xF000_0000	19.4.6/19-13
0xFC03_0040	MII Management Frame Register (MMFR)	32	R/W	Undefined	19.4.7/19-13
0xFC03_0044	MII Speed Control Register (MSCR)	32	R/W	0x0000_0000	19.4.8/19-15
0xFC03_0064	MIB Control/Status Register (MIBC)	32	R/W	0x0000_0000	19.4.9/19-16
0xFC03_0084	Receive Control Register (RCR)	32	R/W	0x05EE_0001	19.4.10/19-16
0xFC03_00C4	Transmit Control Register (TCR)	32	R/W	0x0000_0000	19.4.11/19-17
0xFC03_00E4	Physical Address Low Register (PALR)	32	R/W	Undefined	19.4.12/19-18
0xFC03_00E8	Physical Address High Register (PAUR)	32	R/W	See Section	19.4.13/19-19
0xFC03_00EC	Opcode/Pause Duration (OPD)	32	R/W	See Section	19.4.14/19-19
0xFC03_0118	Descriptor Individual Upper Address Register (IAUR)	32	R/W	Undefined	19.4.15/19-20
0xFC03_011C	Descriptor Individual Lower Address Register (IALR)	32	R/W	Undefined	19.4.16/19-20
0xFC03_0120	Descriptor Group Upper Address Register (GAUR)	32	R/W	Undefined	19.4.17/19-21
0xFC03_0124	Descriptor Group Lower Address Register (GALR)	32	R/W	Undefined	19.4.18/19-21
0xFC03_0144	Transmit FIFO Watermark (TFWR)	32	R/W	0x0000_0000	19.4.19/19-21
0xFC03_014C	FIFO Receive Bound Register (FRBR)	32	R	0x0000_0600	19.4.20/19-22
0xFC03_0150	FIFO Receive FIFO Start Register (FRSR)	32	R	0x0000_0500	19.4.21/19-22
0xFC03_0180	Pointer to Receive Descriptor Ring (ERDSR)	32	R/W	Undefined	19.4.22/19-23
0xFC03_0184	Pointer to Transmit Descriptor Ring (ETDSR)	32	R/W	Undefined	19.4.23/19-23
0xFC03_0188	Maximum Receive Buffer Size (EMRBR)	32	R/W	Undefined	19.4.24/19-24

Table A-11. 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	16.6.1/16-4
0xFC04_4004	eDMA Error Status Register (EDMA_ES)	32	R	0x0000_0000	16.6.2/16-5
0xFC04_400E	eDMA Enable Request Register (EDMA_ERQ)	16	R/W	0x0000	16.6.3/16-8
0xFC04_4016	eDMA Enable Error Interrupt Register (EDMA_EEI)	16	R/W	0x0000	16.6.4/16-9
0xFC04_4018	eDMA Set Enable Request (EDMA_SERQ)	8	W	0x00	16.6.5/16-9
0xFC04_4019	eDMA Clear Enable Request (EDMA_CERQ)	8	W	0x00	16.6.6/16-10
0xFC04_401A	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	8	W	0x00	16.6.7/16-11
0xFC04_401B	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	8	W	0x00	16.6.8/16-11
0xFC04_401C	eDMA Clear Interrupt Request Register (EDMA_CINT)	8	W	0x00	16.6.9/16-12
0xFC04_401D	eDMA Clear Error Register (EDMA_CERR)	8	W	0x00	16.6.10/16-13
0xFC04_401E	eDMA Set START Bit Register (EDMA_SSRT)	8	W	0x00	16.6.11/16-13
0xFC04_401F	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	8	W	0x00	16.6.12/16-14
0xFC04_4026	eDMA Interrupt Request Register (EDMA_INT)	32	R/W	0x0000	16.6.13/16-15
0xFC04_402E	eDMA Error Register (EDMA_ERR)	32	R/W	0x0000	16.6.14/16-15
0xFC04_4100 + hex(n)	eDMA Channel n Priority Register (DCHPRIn) for $n = 0 - 15$	8	R/W	See Section	16.6.15/16-16
0xFC04_5000 + hex($32 \times n$)	Transfer Control Descriptor (TCDn) for $n = 0 - 15$	256	R/W	See Section	16.6.16/16-17

Table A-12. 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	14.2.1/14-4
0xFC04_8004	Interrupt Pending Register Low (IPRL0)	32	R	0x0000_0000	14.2.1/14-4
0xFC04_8008	Interrupt Mask Register High (IMRH0)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_800C	Interrupt Mask Register Low (IMRL0)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_8010	Interrupt Force Register High (INTFRCH0)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_8014	Interrupt Force Register Low (INTFRCL0)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_801A	Interrupt Configuration Register (ICONFIG)	16	R/W	0x0000	14.2.4/14-7
0xFC04_801C	Set Interrupt Mask (SIMR0)	8	W	0x00	14.2.5/14-8
0xFC04_801D	Clear Interrupt Mask (CIMR0)	8	W	0x00	14.2.6/14-9
0xFC04_801E	Current Level Mask (CLMASK)	8	R/W	0x0F	14.2.7/14-9

Table A-12. Interrupt Controller Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC04_801F	Saved Level Mask (SLMASK)	8	R/W	0x0F	14.2.8/14-10
0xFC04_8040 + n (n=0:63)	Interrupt Control Registers (ICR0n)	8	R/W	0x00	14.2.9/14-11
0xFC04_80E0	Software Interrupt Acknowledge (SWIACK0)	8	R	0x00	14.2.10/14-15
0xFC04_80E0 + 4n (n=1:7)	Level n Interrupt Acknowledge Registers (LnIACK0)	8	R	0x18	14.2.10/14-15
Interrupt Controller 1					
0xFC04_C000	Interrupt Pending Register High (IPRH1)	32	R	0x0000_0000	14.2.1/14-4
0xFC04_C004	Interrupt Pending Register Low (IPRL1)	32	R	0x0000_0000	14.2.1/14-4
0xFC04_C008	Interrupt Mask Register High (IMRH1)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_C00C	Interrupt Mask Register Low (IMRL1)	32	R/W	0xFFFF_FFFF	14.2.2/14-5
0xFC04_C010	Interrupt Force Register High (INTFRCH1)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_C014	Interrupt Force Register Low (INTFRCL1)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_C01C	Set Interrupt Mask (SIMR1)	8	W	0x00	14.2.5/14-8
0xFC04_C01D	Clear Interrupt Mask (CIMR1)	8	W	0x00	14.2.6/14-9
0xFC04_C040 + n (n=1:63)	Interrupt Control Registers (ICR1n)	8	R/W	0x00	14.2.7/14-9
0xFC04_C0E0	Software Interrupt Acknowledge (SWIACK1)	8	R	0x00	14.2.8/14-10
0xFC04_C0E0 + 4n (n=1:7)	Level n Interrupt Acknowledge Registers (LnIACK1)	8	R	0x18	14.2.9/14-11
Global IACK Registers					
0xFC05_40E0	Global Software Interrupt Acknowledge (GSWIACK)	8	R	0x00	14.2.10/14-15
0xFC05_40E0 + 4n (n=1:7)	Global Level n Interrupt Acknowledge Registers (GLnIACK)	8	R	0x18	14.2.10/14-15

Table A-13. I^{2}C Module Memory Map

Address	Register	Access	Reset Value	Section/Page
0xFC05_8000	I^{2}C Address Register (I2ADR)	R/W	0x00	32.2.1/32-3
0xFC05_8004	I^{2}C Frequency Divider Register (I2FDR)	R/W	0x00	32.2.2/32-3
0xFC05_8008	I^{2}C Control Register (I2CR)	R/W	0x00	32.2.3/32-4
0xFC05_800C	I^{2}C Status Register (I2SR)	R/W	0x81	32.2.4/32-5
0xFC05_8010	I^{2}C Data I/O Register (I2DR)	R/W	0x00	32.2.5/32-6

Table A-14. QSPI Memory Map

Address[1]	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC05_C000	QSPI Mode Register (QMR)	16	R/W	0x0104	30.3.1/30-3
0xFC05_C004	QSPI Delay Register (QDLYR)	16	R/W	0x0404	30.3.2/30-5
0xFC05_C008	QSPI Wrap Register (QWR)	16	R/W[2]	0x0000	30.3.3/30-6
0xFC05_C00C	QSPI Interrupt Register (QIR)	16	R/W[2]	0x0000	30.3.4/30-6
0xFC05_C010	QSPI Address Register (QAR)	16	R/W[2]	0x0000	30.3.5/30-7
0xFC05_C014	QSPI Data Register (QDR)	16	R/W	0x0000	30.3.6/30-8

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion.

[2] See the register description for special cases. Some bits may be read- or write-only.

Table A-15. UART Module Memory Map

Address UART0 UART1 UART2	Register	Width (bit)	Access	Reset Value	Section/Page
0xFC06_0000 0xFC06_4000 0xFC06_8000	UART Mode Registers[1] (UMR1n), (UMR2n)	8	R/W	0x00	31.3.1/31-5 31.3.2/31-6
0xFC06_0004 0xFC06_4004 0xFC06_8004	UART Status Register (USRn)	8	R	0x00	31.3.3/31-8
	UART Clock Select Register[1] (UCSRn)	8	W	See Section	31.3.4/31-9
0xFC06_0008 0xFC06_4008 0xFC06_8008	UART Command Registers (UCRn)	8	W	0x00	31.3.5/31-9
0xFC06_000C 0xFC06_400C 0xFC06_800C	UART Receive Buffers (URBn)	8	R	0xFF	31.3.6/31-11
	UART Transmit Buffers (UTBn)	8	W	0x00	31.3.7/31-12
0xFC06_0010 0xFC06_4010 0xFC06_8010	UART Input Port Change Register (UIPCRn)	8	R	See Section	31.3.8/31-12
	UART Auxiliary Control Register (UACRn)	8	W	0x00	31.3.9/31-13
0xFC06_0014 0xFC06_4014 0xFC06_8014	UART Interrupt Status Register (UISRn)	8	R	0x00	31.3.10/31-13
	UART Interrupt Mask Register (UIMRn)	8	W	0x00	31.3.10/31-13
0xFC06_0018 0xFC06_4018 0xFC06_8018	UART Baud Rate Generator Register (UBG1n)	8	W[2]	0x00	31.3.11/31-15
0xFC06_001C 0xFC06_401C 0xFC06_801C	UART Baud Rate Generator Register (UBG2n)	8	W[2]	0x00	31.3.11/31-15

Table A-15. UART Module Memory Map (continued)

Address UART0 UART1 UART2	Register	Width (bit)	Access	Reset Value	Section/Page
0xFC06_0034 0xFC06_4034 0xFC06_8034	UART Input Port Register (UIP*n*)	8	R	0xFF	31.3.12/31-15
0xFC06_0038 0xFC06_4038 0xFC06_8038	UART Output Port Bit Set Command Register (UOP1*n*)	8	W[2]	0x00	31.3.13/31-16
0xFC06_003C 0xFC06_403C 0xFC06_803C	UART Output Port Bit Reset Command Register (UOP0*n*)	8	W[2]	0x00	31.3.13/31-16

[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.

Table A-16. DMA Timer Module Memory Map

Address DMA Timer 0 DMA Timer 1 DMA Timer 2 DMA Timer 3	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC07_0000 0xFC07_4000 0xFC07_8000 0xFC07_C000	DMA Timer *n* Mode Register (DTMR*n*)	16	R/W	0x0000	29.2.1/29-3
0xFC07_0002 0xFC07_4002 0xFC07_8002 0xFC07_C002	DMA Timer *n* Extended Mode Register (DTXMR*n*)	8	R/W	0x00	29.2.2/29-4
0xFC07_0003 0xFC07_4003 0xFC07_8003 0xFC07_C003	DMA Timer *n* Event Register (DTER*n*)	8	R/W	0x00	29.2.3/29-5
0xFC07_0004 0xFC07_4004 0xFC07_8004 0xFC07_C004	DMA Timer *n* Reference Register (DTRR*n*)	32	R/W	0xFFFF_FFFF	29.2.4/29-6

Table A-16. DMA Timer Module Memory Map (continued)

Address DMA Timer 0 DMA Timer 1 DMA Timer 2 DMA Timer 3	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC07_0008 0xFC07_4008 0xFC07_8008 0xFC07_C008	DMA Timer *n* Capture Register (DTCR*n*)	32	R/W	0x0000_0000	29.2.5/29-7
0xFC07_000C 0xFC07_400C 0xFC07_800C 0xFC07_C00C	DMA Timer *n* Counter Register (DTCN*n*)	32	R	0x0000_0000	29.2.6/29-8

Table A-17. Programmable Interrupt Timer Modules Memory Map

Address PIT 0 PIT 1 PIT 2 PIT 3	Register	Width (bits)	Access[1]	Reset Value	Section/Page
Supervisor Access Only Registers[2]					
0xFC08_0000 0xFC08_4000 0xFC08_8000 0xFC08_C000	PIT Control and Status Register (PCSR*n*)	16	R/W	0x0000	28.2.1/28-3
0xFC08_0002 0xFC08_4002 0xFC08_8002 0xFC08_C002	PIT Modulus Register (PMR*n*)	16	R/W	0xFFFF	28.2.2/28-5
User/Supervisor Access Registers					
0xFC08_0004 0xFC08_4004 0xFC08_8004 0xFC08_C004	PIT Count Register (PCNTR*n*)	16	R	0xFFFF	28.2.3/28-5

[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.

Table A-18. PWM Memory Map

Address[1,2]	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC09_0020	PWM Enable Register (PWME)	8	R/W	0x00	26.2.1/26-3
0xFC09_0021	PWM Polarity Register (PWMPOL)	8	R/W	0x00	26.2.2/26-4

Table A-18. PWM Memory Map (continued)

Address[1,2]	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC09_0022	PWM Clock Select Register (PWMCLK)	8	R/W	0x00	26.2.3/26-4
0xFC09_0023	PWM Prescale Clock Select Register (PWMPRCLK)	8	R/W	0x00	26.2.4/26-5
0xFC09_0024	PWM Center Align Enable Register (PWMCAE)	8	R/W	0x00	26.2.5/26-6
0xFC09_0025	PWM Control Register (PWMCTL)	8	R/W	0x00	26.2.6/26-6
0xFC09_0028	PWM Scale A Register (PWMSCLA)	8	R/W	0x00	26.2.7/26-7
0xFC09_0029	PWM Scale B Register (PWMSCLB)	8	R/W	0x00	26.2.8/26-8
0xFC09_002C + n n = 0–7	PWM Channel n Counter Register (PWMCNTn)	8	R/W	0x00	26.2.9/26-9
0xFC09_0034 + n n = 0–7	PWM Channel n Period Register (PWMPERn)	8	R/W	0xFF	26.2.10/26-10
0xFC09_003C + n n = 0–7	PWM Channel n Duty Register (PWMDTYn)	8	R/W	0xFF	26.2.11/26-10
0xFC09_0044	PWM Shutdown Register (PWMSDN)	8	R/W	0x00	26.2.12/26-11

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

[2] A 32-bit access to any of these registers results in a bus transfer error (see Section 11.2.7, "SCM Interrupt Status Register (SCMISR)").

Table A-19. 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	15.4.1/15-3
0xFC09_4002	EPORT Data Direction Register (EPDDR)	8	R/W	0x00	15.4.2/15-4
0xFC09_4003	EPORT Interrupt Enable Register (EPIER)	8	R/W	0x00	15.4.3/15-5
Supervisor/User Access Registers					
0xFC09_4004	EPORT Data Register (EPDR)	8	R/W	0xFF	15.4.4/15-5
0xFC09_4005	EPORT Pin Data Register (EPPDR)	8	R	See Section	15.4.5/15-5
0xFC09_4006	EPORT Flag Register (EPFR)	8	R/W	0x00	15.4.6/15-6

[1] User access to supervisor-only address locations have no effect and result in a bus error.

Table A-20. Watchdog Timer Module Memory Map

Address[1]	Register	Width (bits)	Access	Reset Value	Section/Page
	Supervisor Only Access				
0xFC09_8000	Watchdog Control Register (WCR)	16	R/W	0x000F	27.2.1/27-3
0xFC09_8002	Watchdog Modulus Register (WMR)	16	R/W	0xFFFF	27.2.2/27-4
	Supervisor/User Access				
0xFC09_8004	Watchdog Count Register (WCNTR)	16	R	0xFFFF	27.2.3/27-4
0xFC09_8006	Watchdog Service Register (WSR)	16	R/W	0x0000	27.2.4/27-5

[1] Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

Table A-21. 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	9.3.1/9-3
0xFC0A_0008	Reset Configuration Register (RCON)	16	R	0x0001	9.3.2/9-4
0xFC0A_000A	Chip Identification Register (CIR)	16	R	See Section	9.3.3/9-4
0xFC0A_0010	Miscellaneous Control Register (MISCCR)	16	R/W	See Section	9.3.4/9-5
0xFC0A_0012	Clock Divider Register (CDR)	16	R/W	0x0001	9.3.5/9-6
0xFC0A_0014	USB Host Controller Status Register (UHCSR)	16	R/W	0x0000	9.3.6/9-7
0xFC0A_0016	USB On-the-Go Controller Status Register (UOCSR)	16	R/W	0x0010	9.3.7/9-7

[1] User access to supervisor only address locations have no effect and result in a bus error.

Table A-22. Reset Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0000	Reset Control Register (RCR)	8	R/W	0x00	10.3.1/10-2
0xFC0A_0001	Reset Status Register (RSR)	8	R	See Section	10.3.2/10-3

Table A-23. 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	8.2.1/8-2

MCF5329 Reference Manual, Rev 3

Table A-23. Power Management Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC04_002C	Peripheral Power Management Set Register 0 (PPMSR0)	8	W	0x00	8.2.2/8-3
0xFC04_002D	Peripheral Power Management Clear Register 0 (PPMCR0)	8	W	0x00	8.2.3/8-4
0xFC04_002E	Peripheral Power Management Set Register 1 (PPMSR1)	8	W	0x00	8.2.2/8-3
0xFC04_002F	Peripheral Power Management Clear Register 1 (PPMCR1)	8	W	0x00	8.2.3/8-4
0xFC04_0030	Peripheral Power Management High Register 0 (PPMHR0)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC04_0034	Peripheral Power Management Low Register 0 (PPMLR0)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC04_0038	Peripheral Power Management High Register 1 (PPMHR1)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC0A_0007	Low-Power Control Register (LPCR)	8	R/W	0x00	8.2.5/8-7
0xFC0A_0010	Miscellaneous Control Register (MISCCR)[2]	16	R/W	See Section	9.3.4/9-5
0xFC0A_0012	Clock Divider Register (CDR)[2]	16	R/W	0x0001	9.3.5/9-6

[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 9, "Chip Configuration Module (CCM)."

Table A-24. GPIO Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
	Port Output Data Registers				
0xFC0A_4000	PODR_FECH	8	R/W	0xFF	13.3.1/13-14
0xFC0A_4001	PODR_FECL	8	R/W	0xFF	13.3.1/13-14
0xFC0A_4002	PODR_SSI	8	R/W	0x1F	13.3.1/13-14
0xFC0A_4003	PODR_BUSCTL	8	R/W	0x0F	13.3.1/13-14
0xFC0A_4004	PODR_BE	8	R/W	0x0F	13.3.1/13-14
0xFC0A_4005	PODR_CS	8	R/W	0x3E	13.3.1/13-14
0xFC0A_4006	PODR_PWM	8	R/W	0x3C	13.3.1/13-14
0xFC0A_4007	PODR_FECI2C	8	R/W	0x0F	13.3.1/13-14
0xFC0A_4009	PODR_UART	8	R/W	0xFF	13.3.1/13-14
0xFC0A_400A	PODR_QSPI	8	R/W	0x3F	13.3.1/13-14
0xFC0A_400B	PODR_TIMER	8	R/W	0x0F	13.3.1/13-14
0xFC0A_400D	PODR_LCDDATAH	8	R/W	0x03	13.3.1/13-14
0xFC0A_400E	PODR_LCDDATAM	8	R/W	0xFF	13.3.1/13-14
0xFC0A_400F	PODR_LCDDATAL	8	R/W	0xFF	13.3.1/13-14
0xFC0A_4010	PODR_LCDCTLH	8	R/W	0x01	13.3.1/13-14
0xFC0A_4011	PODR_LCDCTLL	8	R/W	0xFF	13.3.1/13-14

Table A-24. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
	Port Data Direction Registers				
0xFC0A_4014	PDDR_FECH	8	R/W	0x00	13.3.2/13-17
0xFC0A_4015	PDDR_FECL	8	R/W	0x00	13.3.2/13-17
0xFC0A_4016	PDDR_SSI	8	R/W	0x00	13.3.2/13-17
0xFC0A_4017	PDDR_BUSCTL	8	R/W	0x00	13.3.2/13-17
0xFC0A_4018	PDDR_BE	8	R/W	0x00	13.3.2/13-17
0xFC0A_4019	PDDR_CS	8	R/W	0x00	13.3.2/13-17
0xFC0A_401A	PDDR_PWM	8	R/W	0x00	13.3.2/13-17
0xFC0A_401B	PDDR_FECI2C	8	R/W	0x00	13.3.2/13-17
0xFC0A_401D	PDDR_UART	8	R/W	0x00	13.3.2/13-17
0xFC0A_401E	PDDR_QSPI	8	R/W	0x00	13.3.2/13-17
0xFC0A_401F	PDDR_TIMER	8	R/W	0x00	13.3.2/13-17
0xFC0A_4021	PDDR_LCDDATAH	8	R/W	0x00	13.3.2/13-17
0xFC0A_4022	PDDR_LCDDATAM	8	R/W	0x00	13.3.2/13-17
0xFC0A_4023	PDDR_LCDDATAL	8	R/W	0x00	13.3.2/13-17
0xFC0A_4024	PDDR_LCDCTLH	8	R/W	0x00	13.3.2/13-17
0xFC0A_4025	PDDR_LCDCTLL	8	R/W	0x00	13.3.2/13-17
	Port Pin Data/Set Data Registers				
0xFC0A_4028	PPDSDR_FECH	8	R/W	See Section	13.3.3/13-19
0xFC0A_4029	PPDSDR_FECL	8	R/W	See Section	13.3.3/13-19
0xFC0A_402A	PPDSDR_SSI	8	R/W	See Section	13.3.3/13-19
0xFC0A_402B	PPDSDR_BUSCTL	8	R/W	See Section	13.3.3/13-19
0xFC0A_402C	PPDSDR_BE	8	R/W	See Section	13.3.3/13-19
0xFC0A_402D	PPDSDR_CS	8	R/W	See Section	13.3.3/13-19
0xFC0A_402E	PPDSDR_PWM	8	R/W	See Section	13.3.3/13-19
0xFC0A_402F	PPDSDR_FECI2C	8	R/W	See Section	13.3.3/13-19
0xFC0A_4031	PPDSDR_UART	8	R/W	See Section	13.3.3/13-19
0xFC0A_4032	PPDSDR_QSPI	8	R/W	See Section	13.3.3/13-19
0xFC0A_4033	PPDSDR_TIMER	8	R/W	See Section	13.3.3/13-19
0xFC0A_4035	PPDSDR_LCDDATAH	8	R/W	See Section	13.3.3/13-19
0xFC0A_4036	PPDSDR_LCDDATAM	8	R/W	See Section	13.3.3/13-19
0xFC0A_4037	PPDSDR_LCDDATAL	8	R/W	See Section	13.3.3/13-19

Table A-24. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4038	PPDSDR_LCDCTLH	8	R/W	See Section	13.3.3/13-19
0xFC0A_4039	PPDSDR_LCDCTLL	8	R/W	See Section	13.3.3/13-19
Port Clear Output Data Registers					
0xFC0A_403C	PCLRR_FECH	8	W	0x00	13.3.4/13-21
0xFC0A_403D	PCLRR_FECL	8	W	0x00	13.3.4/13-21
0xFC0A_403E	PCLRR_SSI	8	W	0x00	13.3.4/13-21
0xFC0A_403F	PCLRR_BUSCTL	8	W	0x00	13.3.4/13-21
0xFC0A_4040	PCLRR_BE	8	W	0x00	13.3.4/13-21
0xFC0A_4041	PCLRR_CS	8	W	0x00	13.3.4/13-21
0xFC0A_4042	PCLRR_PWM	8	W	0x00	13.3.4/13-21
0xFC0A_4043	PCLRR_FECI2C	8	W	0x00	13.3.4/13-21
0xFC0A_4045	PCLRR_UART	8	W	0x00	13.3.4/13-21
0xFC0A_4046	PCLRR_QSPI	8	W	0x00	13.3.4/13-21
0xFC0A_4047	PCLRR_TIMER	8	W	0x00	13.3.4/13-21
0xFC0A_4049	PCLRR_LCDDATAH	8	W	0x00	13.3.4/13-21
0xFC0A_404A	PCLRR_LCDDATAM	8	W	0x00	13.3.4/13-21
0xFC0A_404B	PCLRR_LCDDATAL	8	W	0x00	13.3.4/13-21
0xFC0A_404C	PCLRR_LCDCTLH	8	W	0x00	13.3.4/13-21
0xFC0A_404D	PCLRR_LCDCTLL	8	W	0x00	13.3.4/13-21
Pin Assignment Registers					
0xFC0A_4050	PAR_FEC	8	R/W	0x00	13.3.5.8/13-29
0xFC0A_4051	PAR_PWM	8	R/W	0x00	13.3.5.11/13-31
0xFC0A_4052	PAR_BUSCTL	8	R/W	0xF8	13.3.5.1/13-24
0xFC0A_4053	PAR_FECI2C	8	R/W	0x00	13.3.5.4/13-26
0xFC0A_4054	PAR_BE	8	R/W	0x0F	13.3.5.2/13-24
0xFC0A_4055	PAR_CS	8	R/W	0x3E	13.3.5.3/13-25
0xFC0A_4056	PAR_SSI	8	R/W	0x0000	13.3.5.10/13-30
0xFC0A_4058	PAR_UART	8	R/W	0x0000	13.3.5.7/13-28
0xFC0A_405A	PAR_QSPI	8	R/W	0x0000	13.3.5.5/13-26
0xFC0A_405C	PAR_TIMER	8	R/W	0x00	13.3.5.6/13-27
0xFC0A_405D	PAR_LCDDATA	8	R/W	0x00	13.3.5.12/13-32
0xFC0A_405E	PAR_LCDCTL	16	R/W	0x0000	13.3.5.13/13-33

Table A-24. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4060	PAR_IRQ	16	R/W	0x0000	13.3.5.9/13-30
Mode Select Control Registers					
0xFC0A_4064	MSCR_FLEXBUS	8	R/W	0x3F	13.3.6/13-34
0xFC0A_4065	MSCR_SDRAM	8	R/W	0x3F	13.3.7/13-34
Drive Strength Control Registers					
0xFC0A_4068	DSCR_I2C	8	R/W	See Section	13.3.8/13-35
0xFC0A_4069	DSCR_PWM	8	R/W	See Section	13.3.8/13-35
0xFC0A_406A	DSCR_FEC	8	R/W	See Section	13.3.8/13-35
0xFC0A_406B	DSCR_UART	8	R/W	See Section	13.3.8.1/13-36
0xFC0A_406C	DSCR_QSPI	8	R/W	See Section	13.3.8/13-35
0xFC0A_406D	DSCR_TIMER	8	R/W	See Section	13.3.8/13-35
0xFC0A_406E	DSCR_SSI	8	R/W	See Section	13.3.8/13-35
0xFC0A_406F	DSCR_LCD	8	R/W	See Section	13.3.8/13-35
0xFC0A_4070	DSCR_DEBUG	8	R/W	See Section	13.3.8/13-35
0xFC0A_4071	DSCR_CLKRST	8	R/W	See Section	13.3.8.2/13-37
0xFC0A_4072	DSCR_IRQ	8	R/W	See Section	13.3.8/13-35

Table A-25. Real Time Clock Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_8000	RTC Hours and Minutes Counter Register (RTC_HOURMIN)	32	R/W	Undefined	25.3.1/25-3
0xFC0A_8004	RTC Seconds Counter Register (RTC_SECONDS)	32	R/W	Undefined	25.3.2/25-4
0xFC0A_8008	RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)	32	R/W	0x0000_0000	25.3.3/25-4
0xFC0A_800C	RTC Seconds Alarm Register (RTC_ALRM_SEC)	32	R/W	0x0000_0000	25.3.4/25-5
0xFC0A_8010	RTC Control Register (RTC_CR)	32	R/W	0x0000_0080	25.3.5/25-5
0xFC0A_8014	RTC Interrupt Status Register (RTC_ISR)	32	R/W	0x0000_0000	25.3.6/25-6
0xFC0A_8018	RTC Interrupt Enable Register (RTC_IER)	32	R/W	0x0000_0000	25.3.7/25-7
0xFC0A_801C	Stopwatch Minutes Register (RTC_STPWCH)	32	R/W	0x0000_003F	25.3.8/25-8
0xFC0A_8020	RTC Days Counter Register (RTC_DAYS)	32	R/W	0x0000_0000	25.3.9/25-9
0xFC0A_8024	RTC Days Alarm Register (RTC_ALRM_DAY)	32	R/W	0x0000_0000	25.3.10/25-9

Table A-26. LCD Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_C000	Screen Start Address Register (LCD_SSAR)	32	R/W	0x0000_0000	22.3.1/22-5
0xFC0A_C004	LCD Size Register (LCD_SR)	32	R/W	0x0000_0000	22.3.2/22-5
0xFC0A_C008	LCD Virtual Page Width Register (LCD_VPW)	32	R/W	0x0000_0000	22.3.3/22-5
0xFC0A_C00C	LCD Cursor Position Register (LCD_CPR)	32	R/W	0x0000_0000	22.3.4/22-6
0xFC0A_C010	LCD Cursor Width Height and Blink Register (LCD_CWHB)	32	R/W	0x0101_00FF	22.3.5/22-7
0xFC0A_C014	LCD Color Cursor Mapping Register (LCD_CCMR)	32	R/W	0x0000_0000	22.3.6/22-8
0xFC0A_C018	LCD Panel Configuration Register (LCD_PCR)	32	R/W	0x0000_0000	22.3.7/22-9
0xFC0A_C01C	LCD Horizontal Configuration Register (LCD_HCR)	32	R/W	0x0400_0000	22.3.8/22-12
0xFC0A_C020	LCD Vertical Configuration Register (LCD_VCR)	32	R/W	0x0400_0000	22.3.9/22-12
0xFC0A_C024	LCD Panning Offset Register (LCD_POR)	32	R/W	0x0000_0000	22.3.10/22-13
0xFC0A_C028	LCD Sharp Configuration Register (LCD_SCR)	32	R/W	0x400C_0373	22.3.11/22-14
0xFC0A_C02C	LCD PWM Contrast Control Register (LCD_PCCR)	32	R/W	0x0000_0000	22.3.12/22-16
0xFC0A_C030	LCD DMA Control Register (LCD_DCR)	32	R/W	0x8010_0004	22.3.13/22-16
0xFC0A_C034	LCD Refresh Mode Control Register (LCD_RMCR)	32	R/W	0x0000_0000	22.3.14/22-17
0xFC0A_C038	LCD Interrupt Configuration Register (LCD_ICR)	32	R/W	0x0000_0000	22.3.15/22-18
0xFC0A_C03C	LCD Interrupt Enable Register (LCD_IER)	32	R/W	0x0000_0000	22.3.16/22-19
0xFC0A_C040	LCD Interrupt Status Register (LCD_ISR)	32	R/W	0x0000_0000	22.3.17/22-20
0xFC0A_C050	LCD Graphic Window Start Address Register (LCD_GWSAR)	32	R/W	0x0000_0000	22.3.18/22-22
0xFC0A_C054	LCD Graphic Window Size Register (LCD_GWSR)	32	R/W	0x0000_0000	22.3.19/22-22
0xFC0A_C058	LCD Graphic Window Virtual Page Width Register (LCD_GWVPW)	32	R/W	0x0000_0000	22.3.20/22-23
0xFC0A_C05C	LCD Graphic Window Panning Offset Register (LCD_GWPOR)	32	R/W	0x0000_0000	22.3.21/22-23
0xFC0A_C060	LCD Graphic Window Position Register (LCD_GWPR)	32	R/W	0x0000_0000	22.3.22/22-24
0xFC0A_C064	LCD Graphic Window Control Register (LCD_GWCR)	32	R/W	0x0000_0000	22.3.23/22-24
0xFC0A_C068	LCD Graphic Window DMA Control Register (LCD_GWDCR)	32	R/W	0x8010_0004	22.3.24/22-26
0xFC0A_C800 ... 0xFC0A_CBFC	Background Look-up Table (BGLUT)	32	R/W	—	22.3.25/22-26
0xFC0A_CC00 ... 0xFC0A_CFFC	Graphic Window Look-up Table (GWLUT)	32	R/W	—	22.3.25/22-26

Table A-27. 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	0x0041_FA05	21.3.1.1/21-9
0xFC0B_0004	General Hardware Parameters (HWGENERAL)	N	H/D	32	R	0x0000_07C5	21.3.1.2/21-10
0xFC0B_0008	Host Hardware Parameters (HWHOST)	N	H/D	32	R	0x1002_0001	21.3.1.3/21-11
0xFC0B_000C	Device Hardware Parameters (HWDEVICE)	N	D	32	R	0x0000_0009	21.3.1.4/21-11
0xFC0B_0010	TX Buffer Hardware Parameters (HWTXBUF)	N	H/D	32	R	0x8004_0604	21.3.1.5/21-12
0xFC0B_0014	RX Buffer Hardware Parameters (HWRXBUF)	N	H/D	32	R	0x0000_0404	21.3.1.6/21-12
Capability Registers							
0xFC0B_0100	Host Interface Version Number (HCIVERSION)	Y	H	16	R	0x0100	21.3.2.1/21-13
0xFC0B_0103	Capability Register Length (CAPLENGTH)	Y	H/D	8	R	0x40	21.3.2.2/21-13
0xFC0B_0104	Host Structural Parameters (HCSPARAMS)	Y	H	32	R	0x0001_0011	21.3.2.3/21-14
0xFC0B_0108	Host Capability Parameters (HCCPARAMS)	Y	H	32	R	0x0000_0006	21.3.2.4/21-15
0xFC0B_0122	Device Interface Version Number (DCIVERSION)	N	D	16	R	0x0001	21.3.2.5/21-15
0xFC0B_0124	Device Capability Parameters (DCCPARAMS)	N	D	32	R	0x0000_0184	21.3.2.6/21-16
Operational Registers							
0xFC0B_0140	USB Command (USBCMD)	Y	H/D	32	R/W	0x0008_0000	21.3.3.1/21-17
0xFC0B_0144	USB Status (USBSTS)	Y	H/D	32	R/W	0x0000_0080	21.3.3.2/21-19
0xFC0B_0148	USB Interrupt Enable (USBINTR)	Y	H/D	32	R/W	0x0000_0000	21.3.3.3/21-21
0xFC0B_014C	USB Frame Index (FRINDEX)	Y	H/D	32	R/W	0x0000_0000	21.3.3.4/21-23
0xFC0B_0154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	H	32	R/W	0x0000_0000	21.3.3.5/21-24
0xFC0B_0154	Device Address (DEVICEADDR)	N	D	32	R/W	0x0000_0000	21.3.3.6/21-25
0xFC0B_0158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	H	32	R/W	0x0000_0000	21.3.3.7/21-25
0xFC0B_0158	Address at Endpoint List (EPLISTADDR)	N	D	32	R/W	0x0000_0000	21.3.3.8/21-26
0xFC0B_015C	Host TT Asynchronous Buffer Control (TTCTRL)	N	H	32	R/W	0x0000_0000	21.3.3.9/21-26
0xFC0B_0160	Master Interface Data Burst Size (BURSTSIZE)	N	H/D	32	R/W	0x0000_0404	21.3.3.10/21-27
0xFC0B_0164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	H	32	R/W	0x0000_0000	21.3.3.11/21-27
0xFC0B_0170	ULPI Register Access (ULPI_VIEWPORT)	N	H/D	32	R/W	0x0000_0000	21.3.3.12/21-29
0xFC0B_0180	Configure Flag Register (CONFIGFLAG)	Y	H/D	32	R	0x0000_0001	21.3.3.13/21-31
0xFC0B_0184	Port Status/Control (PORTSC1)	Y	H/D	32	R/W	0xEC00_0004	21.3.3.14/21-31
0xFC0B_01A4	On-The-Go Status and Control (OTGSC)	N	H/D	32	R/W	0x0000_1020	21.3.3.15/21-35
0xFC0B_01A8	USB Mode Register (MODE)	N	H/D	32	R/W	0x0000_0000	21.3.3.16/21-38

MCF5329 Reference Manual, Rev 3

Table A-27. USB On-The-Go Memory Map (continued)

Address	Register	EHCI[1]	H/D[2]	Width (bits)	Access	Reset	Section/Page
0xFC0B_01AC	Endpoint Setup Status Register (EPSETUPSR)	N	D	32	R/W	0x0000_0000	21.3.3.17/21-39
0xFC0B_01B0	Endpoint Initialization (EPPRIME)	N	D	32	R/W	0x0000_0000	21.3.3.18/21-39
0xFC0B_01B4	Endpoint De-initialize (EPFLUSH)	N	D	32	R/W	0x0000_0000	21.3.3.19/21-40
0xFC0B_01B8	Endpoint Status Register (EPSR)	N	D	32	R	0x0000_0000	21.3.3.20/21-40
0xFC0B_01BC	Endpoint Complete (EPCOMPLETE)	N	D	32	R/W	0x0000_0000	21.3.3.21/21-41
0xFC0B_01C0	Endpoint Control Register 0 (EPCR0)	N	D	32	R/W	0x0080_0080	21.3.3.22/21-42
0xFC0B_01C4	Endpoint Control Register 1 (EPCR1)	N	D	32	R/W	0x0000_0000	21.3.3.23/21-43
0xFC0B_01C8	Endpoint Control Register 2 (EPCR2)	N	D	32	R/W	0x0000_0000	21.3.3.23/21-43
0xFC0B_01CC	Endpoint Control Register 3 (EPCR3)	N	D	32	R/W	0x0000_0000	21.3.3.23/21-43

[1] Indicates if the register is present in the EHCI specification.
[2] Indicates if the register is available in host and/or device modes.

Table A-28. USB Host Controller Memory Map

Address	Register	EHCI[1]	Width (bits)	Access	Reset	Section/Page
Module Identification Registers						
0xFC0B_4000	Identification Register (ID)	N	32	R	0x0041_FA05	21.3.1.1/21-9
0xFC0B_4004	General Hardware Parameters (HWGENERAL)	N	32	R	0x0000_02C5	21.3.1.2/21-10
0xFC0B_4008	Host Hardware Parameters (HWHOST)	N	32	R	0x1002_0001	21.3.1.3/21-11
0xFC0B_4010	TX Buffer Hardware Parameters (HWTXBUF)	N	32	R	0x8004_0404	21.3.1.5/21-12
0xFC0B_4014	RX Buffer Hardware Parameters (HWRXBUF)	N	32	R	0x0000_0404	21.3.1.6/21-12
Capability Registers						
0xFC0B_4100	Host Interface Version Number (HCIVERSION)	Y	16	R	0x0100	21.3.2.1/21-13
0xFC0B_4103	Capability Register Length (CAPLENGTH)	Y	8	R	0x40	21.3.2.4/21-15
0xFC0B_4104	Host Structural Parameters (HCSPARAMS)	Y	32	R	0x0001_0011	21.3.2.3/21-14
0xFC0B_4108	Host Capability Parameters (HCCPARAMS)	Y	32	R	0x0000_0006	21.3.2.4/21-15
Operational Registers						
0xFC0B_4140	USB Command (USBCMD)	Y	32	R/W	0x0008_0B00	21.3.3.1/21-17
0xFC0B_4144	USB Status (USBSTS)	Y	32	R/W	0x0000_1000	21.3.3.2/21-19
0xFC0B_4148	USB Interrupt Enable (USBINTR)	Y	32	R/W	0x0000_0000	21.3.3.3/21-21
0xFC0B_414C	USB Frame Index (FRINDEX)	Y	32	R/W	0x0000_0000	21.3.3.4/21-23
0xFC0B_4154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	32	R/W	0x0000_0000	21.3.3.5/21-24

Table A-28. USB Host Controller Memory Map (continued)

Address	Register	EHCI[1]	Width (bits)	Access	Reset	Section/Page
0xFC0B_4158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	32	R/W	0x0000_0000	21.3.3.7/21-25
0xFC0B_415C	Host TT Asynchronous Buffer Control (TTCTRL)	N	32	R/W	0x0000_0000	21.3.3.9/21-26
0xFC0B_4160	Master Interface Data Burst Size (BURSTSIZE)	N	32	R/W	0x0000_0404	21.3.3.10/21-27
0xFC0B_4164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	32	R/W	0x0002_0000	21.3.3.11/21-27
0xFC0B_4180	Configure Flag Register (CONFIGFLAG)	Y	32	R	0x0000_0001	21.3.3.13/21-31
0xFC0B_4184	Port Status/Control (PORTSC1)	Y	32	R/W	0xEC00_0000	21.3.3.14/21-31
0xFC0B_41A8	USB Mode Register (MODE)	N	32	R/W	0x0000_0003	21.3.3.16/21-38

[1] Indicates if the register is present in the EHCI specification.

Table A-29. 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	18.4.1/18-14
0xFC0B_8004	SDRAM Control Register (SDCR)	32	R/W	0x0000_0000	18.4.2/18-15
0xFC0B_8008	SDRAM Configuration Register 1 (SDCFG1)	32	R/W	0x0000_0000	18.4.3/18-17
0xFC0B_800C	SDRAM Configuration Register 2 (SDCFG2)	32	R/W	0x0000_0000	18.4.4/18-19
0xFC0B_8110	SDRAM Chip Select 0 Configuration (SDCS0)	32	R/W	0x0000_0000	18.4.5/18-20
0xFC0B_8114	SDRAM Chip Select 1 Configuration (SDCS1)	32	R/W	0x0000_0000	18.4.5/18-20

Table A-30. 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	24.3.1/24-8
0xFC0B_C004	SSI Transmit Data Register 1 (SSI_TX1)	32	R/W	0x0000_0000	24.3.1/24-8
0xFC0B_C008	SSI Receive Data Register 0 (SSI_RX0)	32	R	0x0000_0000	24.3.4/24-10
0xFC0B_C00C	SSI Receive Data Register 1 (SSI_RX1)	32	R	0x0000_0000	24.3.4/24-10
0xFC0B_C010	SSI Control Register (SSI_CR)	32	R/W	0x0000_0000	24.3.7/24-13
0xFC0B_C014	SSI Interrupt Status Register (SSI_ISR)	32	R	0x0000_3003	24.3.8/24-15
0xFC0B_C018	SSI Interrupt Enable Register (SSI_IER)	32	R/W	0x0000_3003	24.3.9/24-20
0xFC0B_C01C	SSI Transmit Configuration Register (SSI_TCR)	32	R/W	0x0000_0200	24.3.10/24-21
0xFC0B_C020	SSI Receive Configuration Register (SSI_RCR)	32	R/W	0x0000_0200	24.3.11/24-23
0xFC0B_C024	SSI Clock Control Register (SSI_CCR)	32	R/W	0x0004_0000	24.3.12/24-24
0xFC0B_C02C	SSI FIFO Control/Status Register (SSI_FCSR)	32	R/W	0x0081_0081	24.3.13/24-25

Table A-30. SSI Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_C038	SSI AC97 Control Register (SSI_ACR)	32	R/W	0x0000_0000	24.3.14/24-27
0xFC0B_C03C	SSI AC97 Command Address Register (SSI_ACADD)	32	R/W	0x0000_0000	24.3.15/24-28
0xFC0B_C040	SSI AC97 Command Data Register (SSI_ACDAT)	32	R/W	0x0000_0000	24.3.16/24-29
0xFC0B_C044	SSI AC97 Tag Register (SSI_ATAG)	32	R/W	0x0000_0000	24.3.17/24-29
0xFC0B_C048	SSI Transmit Time Slot Mask Register (SSI_TMASK)	32	R/W	0x0000_0000	24.3.18/24-30
0xFC0B_C04C	SSI Receive Time Slot Mask Register (SSI_RMASK)	32	R/W	0x0000_0000	24.3.19/24-30

Table A-31. PLL Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_0000	PLL Output Divider Register (PODR)	8	R/W	0x26	7.2.1/7-6
0xFC0C_0004	PLL Control Register (PCR)	8	R/W	0x00	7.2.2/7-6
0xFC0C_0008	PLL Modulation Divider Register (PMDR)	8	R/W	0x00	7.2.3/7-7
0xFC0C_000C	PLL Feedback Divider Register (PFDR)	8	R/W	0x5A[1]	7.2.4/7-8

[1] With default reset configuration ($\overline{\text{RCON}}$ is negated).

Table A-32. MDHA Module Memory Map

Address	Register	Width (bits)	Access[1]	Reset Value	Section/Page
0xEC08_0000	MDHA Mode Register (MDMR)	32	R/W	0x0000_0000	33.2.1/33-3
0xEC08_0004	MDHA Control Register (MDCR)	32	R/W	0x0000_0000	33.2.2/33-6
0xEC08_0008	MDHA Command Register (MDCMR)	32	W	0x0000_0000	33.2.3/33-7
0xEC08_000C	MDHA Status Register (MDSR)	32	R	0x0000_8408	33.2.4/33-8
0xEC08_0010	MDHA Interrupt Status Registers (MDISR)	32	R	0x0000_0000	33.2.5/33-9
0xEC08_0014	MDHA Interrupt Mask Registers (MDIMR)	32	R/W	0x0000_0000	33.2.5/33-9
0xEC08_001C	MDHA Data Size Register (MDDSR)	32	R/W	0x0000_0000	33.2.6/33-11
0xEC08_0020	MDHA Input FIFO (MDIN)	32	W	0x0000_0000	33.2.7/33-11
0xEC08_0030	MDHA Message Digest A0 Register (MDA0)	32	R/W	0x0123_4567	33.2.8/33-11
0xEC08_0034	MDHA Message Digest B0 Register (MDB0)	32	R/W	0x89AB_CDEF	33.2.8/33-11
0xEC08_0038	MDHA Message Digest C0 Register (MDC0)	32	R/W	0xFEDC_BA98	33.2.8/33-11
0xEC08_003C	MDHA Message Digest D0 Register (MDD0)	32	R/W	0x7654_3210	33.2.8/33-11
0xEC08_0040	MDHA Message Digest E0 Register (MDE0)	32	R/W	0xF0E1_D2C3	33.2.8/33-11
0xEC08_0044	MDHA Message Data Size Register (MDMDS)	32	R/W	0x0000_0000	33.2.9/33-12

Table A-32. MDHA Module Memory Map (continued)

Address	Register	Width (bits)	Access[1]	Reset Value	Section/Page
0xEC08_0070	MDHA Message Digest A1 Register (MDA1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_0074	MDHA Message Digest B1 Register (MDB1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_0078	MDHA Message Digest C1 Register (MDC1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_007C	MDHA Message Digest D1 Register (MDD1)	32	R/W	0x0000_0000	33.2.10/33-12
0xEC08_0080	MDHA Message Digest E1 Register (MDE1)	32	R/W	0x0000_0000	33.2.10/33-12

[1] Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

Table A-33. SKHA Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC08_4000	SKHA Mode Register (SKMR)	32	R/W	0x0000_0000	35.2.1/35-6
0xEC08_4004	SKHA Control Register (SKCR)	32	R/W	0x0000_0000	35.2.2/35-7
0xEC08_4008	SKHA Command Register (SKCMR)	32	R/W	0x0000_0000	35.2.3/35-8
0xEC08_400C	SKHA Status Register (SKSR)	32	R	0x0000_0000	35.2.4/35-9
0xEC08_4010	SKHA Error Status Register (SKESR)	32	R	0x0000_0000	35.2.5/35-10
0xEC08_4014	SKHA Error Status Mask Register (SKEMR)	32	R/W	0x0000_0000	35.2.5/35-10
0xEC08_4018	SKHA Key Size Register (SKKSR)	32	R/W	0x0000_0000	35.2.6/35-12
0xEC08_401C	SKHA Data Size Register (SKDSR)	32	R/W	0x0000_0000	35.2.7/35-12
0xEC08_4020	SKHA Input FIFO (SKIN)	32	R/W	0x0000_0000	35.2.8/35-13
0xEC08_4024	SKHA Output FIFO (SKOUT)	32	R/W	0x0000_0000	35.2.9/35-13
0xEC08_4030	SKHA Key Data Register 1 (SKKDR1)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4034	SKHA Key Data Register 2 (SKKDR2)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4038	SKHA Key Data Register 3 (SKKDR3)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_403C	SKHA Key Data Register 4 (SKKDR4)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4040	SKHA Key Data Register 5 (SKKDR5)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4044	SKHA Key Data Register 6 (SKKDR6)	32	W	0x0000_0000	35.2.10/35-13
0xEC08_4070	SKHA Context 1 (SKC1)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4074	SKHA Context 2 (SKC2)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4078	SKHA Context 3 (SKC3)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_407C	SKHA Context 4 (SKC4)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4080	SKHA Context 5 (SKC5)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4084	SKHA Context 6 (SKC6)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4088	SKHA Context 7 (SKC7)	32	R/W	0x0000_0000	35.2.11/35-14

Table A-33. SKHA Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC08_408C	SKHA Context 8 (SKC8)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4090	SKHA Context 9 (SKC9)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4094	SKHA Context 10 (SKC10)	32	R/W	0x0000_0000	35.2.11/35-14
0xEC08_4098	SKHA Context 11 (SKC11)	32	R/W	0x0000_0000	35.2.11/35-14

Table A-34. RNG Block Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC08_8000	RNG Control Register (RNGCR)	32	R/W	0x0000_0000	34.2.1/34-2
0xEC08_8004	RNG Status Register (RNGSR)	32	R	0x0010_0000	34.2.2/34-3
0xEC08_8008	RNG Entropy Register (RNGER)	32	W	0x0000_0000	34.2.3/34-4
0xEC08_800C	RNG Output FIFO (RNGOUT)	32	R	0x0000_0000	34.2.4/34-4

Appendix B
Revision History

This appendix lists major changes between versions of the MCF5329RM document.

B.1   Changes Between Rev. 2 and Rev. 3

Table B-1. MCF5329RM Rev 2 to Rev. 3 Changes

Chapter	Description
Overview	Corrected MCF53281 column in features list table. This device contains FlexCAN but does not have cryptography accelerators.
Signal Descriptions	In Signal Information and Muxing table, moved MCF53281 label from the MCF5328 column to the MCF5329 column, because this device contains CAN output signals.
	Corrected pinouts in Signal Information and Muxing table for 196 MAPBGA device: Changed D[15:1] entry from "F4–F1, G4–G2..." to "F4–F1, G5–G2..." Changed DSO/TDO entry from "P9" to "N9"
	Updated pinouts for 196 MAPBGA device, MCF5327CVM240 in Signal Information and Muxing table. The following locations are affected: G10–12, H12–14, J11–14, K12–13, L12–13, M12–14, N13. The following signals are affected: USBOTG_VDD, USBHOST_VSS, USBOTG_M, USBOTG_P, USBHOST_M, USBHOST_P, DRAMSEL, PWM3, PWM1, IRQ[7,4,3,2,1], RESET, TDI/DSI, JTAG_EN, TMS/BKPT.
	Removed footnote 2 from the IRQ[7:1] alternate functions USBHOST VBUS_EN, USBHOST VBUS_OC, SSI_MCLK, USB_CLKIN, and SSI_CLKIN signals in Signal Information and Muxing table.
Core	In the figure D0 Hardware Configuration Info, updated information for bit 10
	Changed reset values for VBR from 0x0000_0000 to undefined for the lower reserved bits.
Cache	Added note to ACRn section: "Peripheral space (0xE000_0000-0xFFFF_FFFF) should not be cached. The combination of the CACR defaults and the two ACRn registers must define the non-cacheable attribute for this address space."
Reset controller	Clarified Loss of Lock Reset section that this reset only occurs when in PLL mode.
SCM	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.
	In memory map table, swapped MPR0 and MPR1 register addresses.
	In memory map table, swapped BMT0 and BMT1 register addresses. In BMT register figure, swapped BMT0 and BMT1 register addresses.

Table B-1. MCF5329RM Rev 2 to Rev. 3 Changes (continued)

Chapter	Description
GPIO	Corrected reset values of the PDDR_x registers to 0x00.
	Corrected stem sentence in PAR_UART section.
	In Signal Information and Muxing Table, moved MCF53281 label from the MCF5328 column to the MCF5329 column, because this device contains CAN output signals.
	Corrected pinouts in Signal Information and Muxing Table for 196 MAPBGA device: Changed D[15:1] entry from "F4–F1, G4–G2..." to "F4–F1, G5–G2..." Changed DSO/TDO entry from "P9" to "N9"
	Updated pinouts for 196 MAPBGA device, MCF5327CVM240 in Signal Information and Muxing Table. The following locations are affected: G10–12, H12–14, J11–14, K12–13, L12–13, M12–14, N13. The following signals are affected: USBOTG_VDD, USBHOST_VSS, USBOTG_M, USBOTG_P, USBHOST_M, USBHOST_P, DRAMSEL, PWM3, PWM1, $\overline{IRQ}$[7,4,3,2,1], $\overline{RESET}$, TDI/DSI, JTAG_EN, TMS/$\overline{BKPT}$.
	Removed footnote 2 from the IRQ[7:1] alternate functions USBHOST VBUS_EN, USBHOST VBUS_OC, SSI_MCLK, USB_CLKIN, and SSI_CLKIN signals in Signal Information and Muxing Table.
Interrupt Controller	Removed ICONFIG1 register and added note to this section. Similar to the SLMASK and CLMASK registers, there is only 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 Controller	Removed $\overline{DREQ2}$ signal in DMA request summary table.
	Added external signal timing section.
FlexBus	Changed bit description for CSCRn[RDAH,WRAH] fields
	Updated introduction sentence for figure "Longword-Read Burst from 8-Bit Port 3-1-1-1 (Address Setup and Hold)", added note, added footnote to AH, and added dotted line to $\overline{FB\_TA}$ for internal termination.
	Added notes regarding FlexBus signals tristating between bus cycles throughout. Added footnote to FlexBus Signal Summary table regarding signal directions changing during SDRAM 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
	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.
	Added Connections for External Memory Port Sizes (CSCRn[SBM] = 1) figure

Table B-1. MCF5329RM Rev 2 to Rev. 3 Changes (continued)

Chapter	Description
SDRAM Controller	Added step 3 to initialization/application section: "Configure the slew rate for the SDRAM external pins in the GPIO module."
	Added note in memory map section pointing to the slew rate control register in the GPIO module.
	Added Read Clock Recovery (RCR) Block section.
	Updated SD_DQS signal descriptions.
	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."
FEC	Removed mention of MII_STATUS register in MII Management Frame Register register description section.
	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.
USB Host	Corrected FRINDEX reset value from Undefined to 0x0000_0000.
	Corrected address offsets for the following registers: CAPLENGTH from 0x0100 to 0x0103 HCIVERION from 0x0102 to 0x0100
USB OTG	Changed USB_INTR[NAKE,ULPIE] from read-only to read/write in register figure.
	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 EPCRn[TXR] and EPCRn[RXR] bit descriptions.
	Swapped bit encodings in EPCRn[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."

Table B-1. MCF5329RM Rev 2 to Rev. 3 Changes (continued)

Chapter	Description
LCD Controller	Changed LCD_DCR[BURST] from read-only to read/write.
	Changed note after LCD_DCR figure from using 8-bit burst to 16-bit.
	Changed LCD_GWSAR register from read-only to read/write.
	Changed LCD_GWPR[GWXP] bit description from "...in pixel count (from 0 to GW_XMAX)." to "...in pixel count (from 0 to XMAX)." Changed LCD_GWPR[GWYP] bit description from "...in pixel count (from 0 to GW_YMAX)." to "...in pixel count (from 0 to YMAX)."
	Corrected LCD_GWCR section stem sentence.
	Changed note in LCD_PCR[PCD] bit description to "Set PCD so that the LCD_LSCLK frequency is less than one-third (TFT mode) or one-fourth (CSTN mode) of the system bus clock ($f_{sys/3}$) frequency. Otherwise, the line data (LCD_D) is incorrect."
FlexCAN	Corrected step 3 in Receive Process section from "Control/status word to mark the Rx MB as active and empty (CODE = 1000)" to "Control/status word to mark the Rx MB as active and empty (CODE = 0100)"
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.
	Corrected RTC_CR[SWR] bit description
PWM	Changed bit description of PWME[PWME7].
PIT	Corrected PCSR*n* address in memory map table from 0xFC07_8000 to 0xFC08_8000
	Corrected PIT timeout period equation.
DMA Timers	Clarified DTMR*n*[PS] field description.
	In the section **Features**, updated the maximum timeout period information
	In the table **DTMR*n* Field Descriptions**, added the sentence "Avoid setting CLK when RST is set..." to the CLK row description
QSPI	Updated the **Introduction** section's text
	In the section **External Signal Description**, updated the final paragraph
	In the table **QDLYR Field Descriptions**, updated the 15 SPE row
	Updated the table **QDR Field Descriptions**
	Updated the second paragraph of the section **QSPI RAM**
	Updated the first paragraph of the section **Receive RAM**
	Reserved QMR[14] bit (previously DOHIE bit).
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]."
RNG	Added note in overview section.
	Added link to NIST SP800-90 in the overview section.

MCF5329 Reference Manual, Rev 3

Table B-1. MCF5329RM Rev 2 to Rev. 3 Changes (continued)

Chapter	Description
Debug Module	Added CSR[1:0] bits.
	Added note to CSR[PCD] bit description.
	Rearranged sections
JTAG	Changed reset value of IDCODE[DC] to see note, and added note that this value varies, depending on design center location.
Memory Map	Corrected PWM base memory map from 0xFC09_000 to 0xFC09_0020

B.2   Changes Between Rev. 1 and Rev. 2

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes

Chapter	Description
Overview	Added MCF53281 device to family configuration table.
	Added VoIP system solution to features list and added VoIP section.
Signal Descriptions	Table 2-1/Page 2-2: Change D[31:0] signal direction from output (O) to input/output (I/O).
	Table 2-1/Page 2-2: Change MCF5327 196 MAPBGA ball locations for the following signals:

Signal Name	MCF5327 196 MAPBGA	
	Old Incorrect Location	New Correct Location
$\overline{\text{RCON}}$	N7	M7
D1	M6	L6
D0	N6	M6
$\overline{\text{OE}}$	L7	P6
R/$\overline{\text{W}}$	P6	N6
SD_DQS2	L6	K6
PSTCLK	M7	L7
DDATA[3:0]	P7, L8, M8, N8	N7, P7, L8, M8
PST[3:0]	P8, L9, M9, N9	N8, P8, L9, M9
EVDD	E6, E7, F5–F7, H9, J9, J9, K8, K9	E6, E7, F5–F7, H9, J9, J9, K8, K9, K11
IVDD	E5, K5, K10	E5, K5, K10, J10
SD_VDD	E8, E9, F8–F10, J6, K6, J7, K7	E8, E9, F8–F10, J5–J7, K7

	Description
	Table 2-10/Page 2-13: Change LCD_SCLK to LCD_LSCLK to be consistent with the rest of the reference manual.
	Table 2-16/Page 2-17: Change USBOTG_PU_EN to an output and change the description to the following: Enables an external pull-up on the USBOTG_DP line. This signal is controlled by the UOCSR[BVLD] bit.

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description
Core	Table 3-1/Page 3-5: Change PC's Reset Value entry to "Contents of Location 0x0000_0004" and Written with MOVEC entry to "No". Table 3-1/Page 3-5: Change OTHER_A7's Reset Value entry to "Contents of Location 0x0000_0000".
	Figure 3-5/Page 3-7: Change "Access: User read-only" to "Access: read/write". Change CCR[4:0] to read/write.
	Table 3-2/Page 3-8: Remove last sentence in C bit field description.
	Figure 3-8/Page 3-9: Change SR[4:0] to read/write.
	Section 3.4/Page 3-11: Change last bullet to "Use of separate system stack pointers for user and supervisor modes"
	Section 3.4/Page 3-11: Change last sentence in step #2 to "The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address."
EMAC	Changed all mov instructions to move.
	Figure 4-4/Page 4-5: Change MACSR[3:0] to R/W
	Figure 4-5/Page 4-10: Change upper 16 bits of the MASK register to read-only, with a read and reset value of 1
	Equation 4-3/Page 4-13: Add minus sign to the exponent so that it is "$-(i + 1 - N)$".
Cache	Rearranged sections for consistency.
	Table 5-2/Page 5-4: Change reset value of ACR0, ACR1 to "See Section" because some of the bits are defined after reset.
	Section 5.3.2/Page 5-6: Add the following note: **NOTE** Peripheral space (0xE000_0000-0xFFFF_FFFF) should not be cached. The combination of the CACR defaults and the two ACR*n* registers must define the non-cacheable attribute for this address space.
	Section 5.4.6/Page 5-14: Change first sentence from "Ways 0 and 1 of the data cache can be locked by setting CACR[DHLCK]; likewise, ways 0 and 1 of the instruction cache can be locked by setting CACR[IHLCK]." to ""Ways 0 and 1 of the cache can be locked by setting CACR[HLCK]."
	Figure 5-8/Page 5-15: Change sentence near top of figure from "B) CACR[DHLCK] is set, locking ways 0 and 1." to "B) CACR[HLCK] is set, locking ways 0 and 1."
PLL	Remove "the user must wait for the PLL to lock before continuing with code execution." from third paragraph in the Limp Mode section.
Power	Table 8-9/Page 8-8: Added the following note to the LPCR[FWKUP] (fast wake-up) bit description: **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.
	Removed last sentence in first paragraph of Limp Mode section regarding switching from limp to normal mode.
CCM	Section 9.3.5/Page 9-6: The CDR[SSIDIV] field is a 6-bit field. Expand the field in the figure and bit description table from bits 3–0 to bits 5–0
	Table 9-5/Page 9-5: Replace instances of "HCLK" in the LCDCHEN bit description to "internal clock".
SCM	Section 11.2.8/Page 11-10: Combine all BCR[7:0] fields into a single slave burst enable field, SBE. The only valid values for this field are 0x00 and 0xFF. All other values are reserved.
	Added last sentence to first paragraph in PACR*x* section, "At reset the SCM (PACR0) does not allow access from untrusted masters, while the other peripherals do."

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description
Crossbar Switch	Corrected first sentence in XBS_PRS*n*[M3] bit description from "Master 5 (FEC1) priority." to "Master 3 (FEC1) priority."
	Corrected last part of XBS_CRS*n*[RO] = 1 bit setting from "(attempted writes have no effect and result in a bus error response)." to "(attempted writes have no effect on the registers and result in a bus error response)."
	Corrected XBS_CRS*n*[PARK] bit description from "Park. Determines which master port this slave port parks on when..." to "Park. Determines which master port the current slave port parks on when..."
GPIO	Table 13-1/Page 13-4: Change D[31:0] signal direction from output (O) to input/output (I/O).
	Table 13-1/Page 13-4: Change MCF5327 196 MAPBGA ball locations for the following signals:

Signal Name	MCF5327 196 MAPBGA	
	Old Incorrect Location	**New Correct Location**
$\overline{\text{RCON}}$	N7	M7
D1	M6	L6
D0	N6	M6
$\overline{\text{OE}}$	L7	P6
R/$\overline{\text{W}}$	P6	N6
SD_DQS2	L6	K6
PSTCLK	M7	L7
DDATA[3:0]	P7, L8, M8, N8	N7, P7, L8, M8
PST[3:0]	P8, L9, M9, N9	N8, P8, L9, M9
EVDD	E6, E7, F5–F7, H9, J9, J9, K8, K9	E6, E7, F5–F7, H9, J9, J9, K8, K9, K11
IVDD	E5, K5, K10	E5, K5, K10, J10
SD_VDD	E8, E9, F8–F10, J6, K6, J7, K7	E8, E9, F8–F10, J5–J7, K7

Chapter	Description
	Corrected DSCR_MISC field description table title. Previously mislabeled as DSCR_I2C.
Interrupt Controller	Marked CLMASK values 0x8–0xE as reserved in CLMASKfield description table.
	Added ICR000, ICR100, ICR200 to register addresses and section heading in ICR*n* section.
	Added verb "exists" to notes in CLMASK and SLMASK sections.
DMA Controller	Added second paragraph to Modes of Operation, Normal Mode section.
	Added note to TCD*n*_CSR[BWC] field description.

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description
FlexBus	Rearranged sections for consistency.
	Section 17.4.4 & 17.4.5: In figure 17-8 through figure 17-33 add ADDR[31:0] label to first cycle of data signals.
	Section 17.4.4.1/Page 17-12: Change last note on page from 　　　　"The processor drives the data lines during the first clock cycle of the 　　　　transfer. However, this should be ignored by the connected device." to: 　　　　"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 feature beneficial."
	Figure 17-27/Page 17-24: Remove internal termination dashed lines for $\overline{FB\_CS}$, $\overline{FB\_BE/BWE}$, and $\overline{FB\_OE}$ signals.
	Figure 17-31/Page 17-26: Remove internal termination dashed lines for $\overline{FB\_CS}$, $\overline{FB\_BE/BWE}$, and $\overline{FB\_OE}$ signals.
	Figure 17-33/Page 17-27: Remove internal termination dashed lines for $\overline{FB\_CS}$, $\overline{FB\_BE/BWE}$, and $\overline{FB\_OE}$ signals.
SDRAM Controller	Changed SD_VREF signal description.
	Clarified series termination and SD_CLK termination in DDR layout considerations.
	Clarified DDR SDRAM termination circuit example figure.
	Corrected typos in SDCR[DQS_OE] bit description: from DSQ to DQS.
	Table 18-1/Page 18-5: In the SD_DQS table entry add the following note: **Note:** 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.
	Figure 18-2/Page 18-11: Replace figure with Figure 1 from AN2982 "System Design Using the ColdFire MCF5208 Split Bus Architecture" found at http://www.freescale.com/coldfire because it is more thorough.
	Figure 18-2/Page 18-12: SD_D[31:0], DQ[31:0], SD_DQS[3:0], and DQS[3:0] should be SD_D[31:16], DQ[31:16], SD_DQS[3:2], and DQS[3:2], respectively, as the device does not support a 32-bit DDR bus. Replace figure with Figure 2 from AN2982 "System Design Using the ColdFire MCF5208 Split Bus Architecture" found at http://www.freescale.com/coldfire because it is more thorough.
	Section 18.3.5/Page 18-13: Remove this entire section, as the device does not support a 32-bit wide DDR bus.
	Table 18-8/Page 18-17: DQS_OE field should match the corresponding register diagram and be only 2 bits wide at locations 11–10. Change third sentence from "The DSQ_OE[3] bit enables SD_DQS3 and the DSQ_OE[2] bit enables SD_DQS2, and so on." to ""The DSQ_OE[1] bit enables SD_DQS3 and the DSQ_OE[0] bit enables SD_DQS2." Consequently, the reserved bit field currently at location 7–3 should be extended to bits 9–3.

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description

SDRAM Controller (continued)	Table 18-9/Page 18-19: Correct equations and examples in SDCFG1[ACT2RW, PRE2ACT, REF2ACT] fied descriptions. Change ACT2RW to the following and change PRE2ACT and REF2ACT similarly.
	
	"Suggested value = (t_{RCD} x f_{SD_CLK}) - 1 (Round up to nearest integer)
	Example:
	  If t_{RCD} = 20ns and f_{SD_CLK} = 99 MHz
	  Suggested value = (20ns x 99 MHz) - 1 = 0.98; round to 1."

	Section 18.4.5/Page 18-20: Add the following note: **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 there is no high level bus monitor on the device, a reset is the only way to exit the error condition.
FEC	Table 19-2/Page 19-6: Correct MIB block counters end address to 0xFC03_02FF.
	Corrected TCR[FEDN] bit to TCR[FDEN] in TCR section.
	Corrected MIB memory boundarys in FEC module memory map table.
	MII management frame registers section: Reworded sentence from "If the MSCRn register is written to a non-zero value in the case of writing to MMFRn when MSCRn equals 0, an MII frame is generated with the data previously written to the MMFRn." to "If MSCR is cleared while MMFR is written and then MSCR is written with a non-zero value, an MII frame is generated with the data previously written to the MMFR. "
	PAURn register figure, TYPE field change to read-only.
	Table 19-3/Page 19-6: Correct ECR reset value from 0xF000_0002 to 0xF000_0000.
	Table 19-3/Page 19-7: Correct register name typo in FEC memory map at address 0xFC03_0124. This should be the Descriptor Group Lower Address Register (GALR).
	Table 19-4/Page 19-8: Add RMON_R_DROP to the MIB counter memory map at address 0xFC03_0280 with a description of Count of frames not counted correctly.
	Figure 19-6/Page 19-13: Correct ECR reset value from 0xF000_0002 to 0xF000_0000.
FEC (continued)	Section 19.4.7/Page 19-35: Add the following subsection entitled "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 will fetch from memory a BD that has already been processed but not yet written back (that is, it is read a second time with the R bit remaining 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.

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description
USB On-the-Go	Clean-up and reorganization of chapter.
	Removed Host Data Structures section as these are covered in the EHCI specification.
	Table 21-2/Page 21-6: Change USBOTG_PU_EN entry to the following. | Signal	I/O	Description
---	---	---
USBOTG_PU_EN	O	Enables an external pull-up on the USBOTG_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	
	Section 21.3.3.1/Page 21-18: Move USBCMD[ATDTW] from bit location 14 to 12 in the register figure and bit description table. Bit 14 is reserved and must be cleared.
	Table 21-19/Page 21-23: Delete last sentence in UEI bit description.
	Table 21-19/Page 21-23: Add the following sentence to the end of the PCI bit description: "The device controller detects resume signaling only."
USB On-the-Go (continued)	Table 21-43/Page 21-49: Change ZLT bit description to the following: "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 is retired. 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."

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description
LCD Controller	Change LCD_SCLK to LCD_LSCLK throughout to be consistent with rest of the reference manual.
	Table 22-5/Page 22-5: Add the following note to LCD_SR[XMAX] bit field: "The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x32."
	Table 22-5/Page 22-5: Add the following note to LCD_SR[YMAX] bit field: "The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x258."
	Table 22-22/Page 22-22: Add the following note to LCD_GWSR[GW_XMAX] bit field: "The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x32."
	Table 22-22/Page 22-22: Add the following note to LCD_GWSR[GW_YMAX] bit field: "The maximum supported panel size is 800x600 pixels. Therefore the maximum value for this bit field is 0x258."
	Section 22.3.23/Page 22-24: Change note to: The graphics window can only be enabled while the clock to the LCD controller is disabled (MISCCR[LCDCHEN] is cleared). See Chapter 9, "Chip Configuration Module (CCM)," for details on this bit. The graphics window can be disabled at any time.
	Table 22-10/Page 22-10: In PCD bit description change the note to: **Note:** The value of PCD must be set such that the LCD_LSCLK frequency is at least one-third (TFT mode) or one-fourth (CSTN mode) of the peripheral bus clock ($f_{sys/3}$) frequency. Otherwise, the line data (LCD_D) is incorrect.
FlexCAN	Corrected Interrupt section. There are 19 total FlexCAN interrupts; added 16 individual interrupts per MB.
SSI	Changed FIFO size from 8x24 to 8x32.
	Changed ACDAT from 19 bits to 20 bits wide (from SSI_ACDAT[18:0] to SSI_ACDAT[19:0])
	Figure 24-15/Page 24-15: Correct register name in top of register figure from SISRR to SSI_ISR.
	Figure 24-18/Page 24-20: Add R/W RXDIR bit to bit location 5 in SSI_RCR register.
SSI (continued)	Table 24-11/Page 24-21: Add RXDIR bit to bit location 5 with the following description: Gated clock enable. In synchronous mode, this bit enables gated clock mode. 0 Gated clock mode disabled. 1 Gated clock mode enabled.
Real Time Clock	Rescind errata regarding 2HZ/2SEC bitfield name. The interrupt is, in fact, a 2 Hz interrupt.
	Clarified Modes of operation section.
PWM	Corrected the numerical values in the left-aligned and center-aligned examples.
	Section 26.2/Page 26-2: Previous errata from revision 0 has crept in: Add a 0x20 offset to all PWM register addresses in the memory map table. Register addresses should be from 0xFC09_0020 to 0xFC09_0044. Also fix address for the PWMSDN register description to 0xFC09_0044.
Watchdog Timer	Figure 27-1/Page 27-2: Change 8192 divider to 4096.
	Section 27.2.3/Page 27-4: Change 8192 multiplier in equation 20-1 and text below it to 4096. As a result the maximum timeout frequency changes from 6.71 to 3.36 seconds.
PITs	Figure 28-4/Page 28-5: Remove "IPSBAR Offset" from PCNTRn register diagram.
DMA Timers	Corrected DTRRn reset value in timer memory map from 0x1111_1111 to 0xFFFF_FFFF.
QSPI	Removed mention of QSPI_CS3 throughout as it is not available on this device.

Table B-2. MCF5329RM Rev 1 to Rev. 2 Changes (continued)

Chapter	Description
UART	Receiver timing diagram: replaced U*n*TXD with U*n*RXD and TXRTS with RXRTS
	Remote loopback figure: changed U*n*TXD label to output
	Multidrop mode timing diagram, master station: changed UMR1*n*[PT]=2 to UMR1*n*[PT]=1, peripheral station: deleted extraneous second instance of UMR1*n*[PM]=11
	UCSRn register diagram: added note to UCSR[TCS,RCS] reset value
UART	Removed unnecessary introduction text to external signal description section.
	Table 31-6 / Page 31-9: Changed "DTIN" to "DT*n*IN" (to maintain consistent signal names throughout chapter).
	Section 31.4.5.2 / Page 31-26: Changed "...complete normally without exception processing..." to "...complete normally without an error termination...".
I^{2}C	Rearranged and renamed sections to be consistent with rest of the reference manual.
Debug Chapter	CPU halt section: Combined second and third sentences in bullet #2 from: "This type of halt is always first made pending in the processor. Next, the processor samples for pending halt and interrupt conditions once per instruction." to: "This type of halt is always first marked as pending in the pocessor, which samples for pending halt and interrupt conditions once per instruction."
	Changed reset value for PBR1–3 in debug memory map table from 0x0000_0000 to See Section.
	Changed reset value for PBR1–3 in PBR register diagram: last 3 bits of register are 000.
	Clarified last sentence of first paragraph in memory map section regarding quiscent DSCLK during WDEBUG.
JTAG	Added MCF53281 IDCODE values.
	Figure 37-3/Page 37-4: Updated the IDCODE register figure to indicate that the reset values for PRN and PIN are device-dependent.
Appendix B	Added revision history appendix. This chapter replaces the separate MCF5329RMAD document.

B.3 Changes Between Rev. 0.1 and Rev. 1

Table B-3. MCF5329RM Rev 0.1 to Rev. 1 Changes

Location	Description
Table 2-1/Page 2-2	Add "Voltage Domain" column indicated the domain for each signal. The USB signals are USB_VDD, FlexBus and SDRAM signals are SD_VDD, while all the rest are EVDD.
Table 2-1/Page 2-2	Add the following footnote to the SD_BA[1:0], SD_A[13:11], SD_A[9:0], and $\overline{SD\_DQM}$[3:0] signals: The SDRAM functions of these signals are not programmable by the user. They are dynamically switched by the processor when accessing SDRAM memory space and are included here for completeness.
Table 2-1/Page 2-4	Remove FEC signals as alternate 1 function of the LCD pins. The FEC signals are available as primary functions on other pins.
Figure 2-1/Page 2-2	Remove FEC signals as alternate 1 function of the LCD pins. The FEC signals are available as primary functions on other pins.
Table 2-1/Page 2-6	Remove FEC signals as alternate 1 function of the LCD pins. The FEC signals are available as primary functions on other pins.
Table 2-1/Page 2-7	Change USBHOST_VSS entry to USB_VSS and USBOTG_VDD entry to USB_VDD.

Table B-3. MCF5329RM Rev 0.1 to Rev. 1 Changes (continued)

Location	Description
Table 2-6/Page 2-10	Correct description of the $\overline{BE}/\overline{BWE}n$ signals: Remove "SRAM and" from second sentence of first paragraph, Remove SDRAM paragraph and add reference to Table 2-7.
Table 2-7/Page 2-11	Add the following to the SD_DQM[3:0] entry description: "These pins are multiplexed with the $\overline{BE}/\overline{BWE}n$ pins. The SD_DQMn should be connected to individual SDRAM DQM signals. Note that most SDRAMs associate DQM3 with the MSB, in which case SD_DQM3 should be connected to the SDRAM's DQM3 input."
Section 3.2.5/Page 3-7	Change bit 7 of the CCR register from a reserved bit to a read/write bit labeled P. In corresponding bit description table, add a row for the P bit with the below description: Branch prediction bit. Alters the static prediction algorithm used by the branch acceleration logic in the IFP on forward conditional branches. 0 Predicted as not taken. 1 Predicted as taken.
Figure 3-9/Page 3-10	Change bit 7 from a reserved bit to a read/write bit labeled P.
Figure 5.2/Page 5-2	Change TAG definition label from "TAG—21-bit address tag" to "TAG—20-bit address tag".
Table 9-10/Page 9-10	Correct boot port size (D[4:3]) settings. They should match Table 9-3. <table><tr><th>D[4:3]</th><th>Boot Device</th></tr><tr><td>00</td><td>External with 32-bit port[3] (default)</td></tr><tr><td>01</td><td>External with 16-bit port</td></tr><tr><td>10</td><td>External with 8-bit port</td></tr><tr><td>11</td><td>External with 32-bit port</td></tr></table>
Table 11-1/Page 11-2	Change reset values of MPR0 to 0x7777_7777. Change reset value of PACRA to 0x5444_4444. Change reset values of the other PACRs to 0x4444_4444. Change CFDTR register address from 0xFC04_0078 to 0xFC04_007C.
Figure 11-1/Page 11-3	Change reset value of MPR0 from 0x7000_0007 to 0x7777_7777.
Section 11.2.3/Page 11-4	Change reset values of PACRA to 0x5444_4444 and of the rest of the PACRs to 0x4444_4444.
Section 11.2.4/Page 11-7	Change next to last sentence in second paragraph from "...an interrupt to the interrupt controller is generated if the CFLOC[ECFEI] bit is set." to ""...an interrupt to the interrupt controller is generated if the CFIER[ECFEI] bit is set."
Section 11.2.8/Page 11-10	Change last sentence in first paragraph from "There is a enable bit..." to "There is an enable bit..."
Figure 11-22/Page 11-14	Change CFDTR register address from 0xFC04_0078 to 0xFC04_007C.

Table B-3. MCF5329RM Rev 0.1 to Rev. 1 Changes (continued)

Location	Description
Table 12-1/Page 12-2	For the slave modules add a column for each slave's address range, as well as two footnotes as shown below: **Table 12-1. Cross-bar Switch Master/Slave Assignments** {{TABLE_A}}
Section 12.1/Page 12-2	Add the following note below the above added table. **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, e.g., ADDR[31]=0 identifies the cacheable space.

Table 12-1 content:

Master Modules	
Cross-bar Port	**Module**
Master 0 (M0)	ColdFire Core
Master 1 (M1)	eDMA Controller
Master 2 (M2)	Fast Ethernet Controller
Master 4 (M4)	LCD Controller
Master 5 (M5)	USB Host
Master 6 (M6)	USB On-the-Go
Master 7 (M7)	Reserved for Factory Test

Slave Modules		
Cross-bar Port	**Module**	**Address Range**[1]
Slave 1 (S1)	Flexbus SDRAM Controller	0x0000_0000–0x3FFF_FFFF & 0xC000_0000–0xDFFF_FFFF 0x4000_0000–0x7FFF_FFFF
Slave 4 (S4)	Internal SRAM Backdoor	0x8000_0000–0x8FFF_FFFF
Slave 6 (S6)	Cryptography Modules (RNG, SKHA, MDHA)	0xE000_0000–0xEFFF_FFFF[2]
Slave 7 (S7)	Other On-chip 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 should not be accessed.

Table B-3. MCF5329RM Rev 0.1 to Rev. 1 Changes (continued)

Location	Description
Table 13-1/Page 13-2	Add "Voltage Domain" column indicated the domain for each signal. The USB signals are USB_VDD, FlexBus and SDRAM signals are SD_VDD, while all the rest are EVDD.
Table 13-1/Page 13-2	Add the following footnote to the SD_BA[1:0], SD_A[13:11], SD_A[9:0], and $\overline{SD\_DQM}$[3:0] signals: The SDRAM functions of these signals are not programmable by the user. They are dynamically switched by the processor when accessing SDRAM memory space and are included here for completeness.
Figure 13-1/Page 13-2	Remove FEC signals as alternate 1 function of the LCD pins. The FEC signals are available as primary functions on other pins.
Table 13-1/Page 13-6	Remove FEC signals as alternate 1 function of the LCD pins. The FEC signals are available as primary functions on other pins.
Table 13-1/Page 13-10	Change USBHOST_VSS entry to USB_VSS and USBOTG_VDD entry to USB_VDD.
Table 13-2/Page 13-11	Remove FEC signal rows from table as they are not available as alternate functions on the LCD pins.
Figure 13-36/Page 13-25	Change figure title to "Chip Select Pin Assignment Register (PAR_CS)"
Table 13-10/Page 13-25	Correct table title to "PAR_CS Field Descriptions" Correct bit field names for bit 5 and bit 4. Bit 5 field should be PAR_CS5 and bit field 4 should be PAR_CS4. The corresponding bit descriptions are correct.
Table 13-19/Page 13-32	For PAR_LD15_8 and PAR_LD7_0 entries in table, change "FEC functions" to "Reserved" as the FEC signals are not available as alternate functions on the LCD pins.
Table 14-15/Page 14-12	Correct source #25, "Flag clearing mechanism" entry from "Write CWIR[CWIC]=1" to "Write SCMISR[CWIC]=1".
Table 14-16/Page 14-14	Replace source #2 entry, "Wake-up interrupt" to "Not Used" as the FlexCAN wake-up interrupt is not supported on this device.
Figure 15-3/Page 15-4	Change register diagram from 16-bit to 8-bit as shown below: 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 | 0 Reset: 0 0 0 0 0 0 0 0 **Figure 15-3. EPORT Data Direction Register (EPDDR)**
Throughout Chapter 17	In all timing diagrams within the FlexBus chapter, the address and data signals stop driving the bus one clock cycle sooner than shown. Because of this, in the diagrams illustrating address hold, the address hold state is swapped with the S3 state.
Figure 17-28/Page 17-23	Insert a single clock cycle for address setup at the beginning of each data transfer, between S0 and S1 states.
Figure 17-29/Page 17-24	Insert a single clock cycle for address setup at the beginning of each data transfer, between S0 and S1 states.
Section 18.5.2/Page 18-23	Remove "Perform a single read/write to any of the SDRAM chip select address spaces to issue the command." sentence from steps 5 and 9 of Section 18.6, "Initialization/Application Information." This is not recommended or necessary.
Table 19-2/Page 19-5	Change reset value of ECR register from 0x0000_0000 to 0xF000_0002.

MCF5329 Reference Manual, Rev 3

Table B-3. MCF5329RM Rev 0.1 to Rev. 1 Changes (continued)

Location	Description
Section 19.2.3/Page 19-6	Change second sentence in first paragraph from "These fall in the 0xFC03_0200-0xFC03_03FF address offset range." to "These fall in the 0xFC03_0200-0xFC03_02FF address range."
Figure 19-6/Page 19-12	Change reset value of ECR register from 0x0000_0000 to 0xF000_0002.
Table 19-11/Page 19-15	Change 80MHz entry for MSCR[MII_SPEED] from 0xF to 0x10.
Section 21.3.3.15/Page 21-37	Change bit 2 in the USBMODE register from a reserved bit to a read/write bit labeled ES. In corresponding bit description table, change bit 2 to the ES bit with the below description: 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. For proper operation, this bit must be set for this ColdFire device.
Table 36-5/Page 36-8	Clarify in CSR field descriptions that the read-only bits can only be accessed via the BDM port and not read via the processor. The CSR is supervisor write-only from the processor.
Appendix A	Added memory map appendix.

B.4 Changes Between Rev. 0 and Rev. 0.1

Table B-4. MCF5329RM Rev. 0 to Rev. 0.1 Changes

Location	Description
Table 2-1/Page 2-1	The table is incorrect and references the MCF5373 family of devices. The correct table is shown in Table 13-1.
Table 2-2/Page 2-7	Replace the table with the following. Notice the addition of the D0 entry:

Table 2-2. Internal Pull-up/down Resistors

Pin Name	Pull-Up	Pull-Down	Comment
RESET	x		Always, except JTAG mode
TEST		x	Always, except JTAG mode
RCON	x		Always, except JTAG mode
XTAL	x		When not in crystal oscillator mode (intended for factory test)
IRQ[7:1]	x		
TA	x		Only when used as TA
D0	x		During reset only
QSPI_DOUT	x		I^2C mode only (I2C_SDA)
QSPI_CLK	x		I^2C mode only (I2C_SCL)
FEC_MDIO	x		I^2C mode only (I2C_SDA)
FEC_MDC	x		I^2C mode only (I2C_SCL)
I2C_SDA	x		I^2C mode only (I2C_SDA)
I2C_SCL	x		I^2C mode only (I2C_SCL)
DT0IN	x		When used as DREQ0
U1RXD	x	x	When used as SSI_RXD, configured by the MISCCR register in the CCM
U1TXD	x	x	When used as SSI_TXD, configured by the MISCCR register in the CCM
SSI_RXD	x	x	SSI mode only. Configured by the MISCCR register in the CCM
SSI_TXD	x	x	SSI mode only. Configured by the MISCCR register in the CCM
JTAG_EN		x	
TDI	x		JTAG mode only
TMS	x		JTAG mode only
TRST	x		JTAG mode only
TCLK	x		JTAG mode only

Table B-4. MCF5329RM Rev. 0 to Rev. 0.1 Changes (continued)

Location	Description
Section 2.3	The LCD controller signals section is missing. Insert the following section & table: **2.3.8 LCD Controller Signals** Table 23-9 describes the LCD controller signals. **Table 23-9. LCD Signals** _(see table below)_
Section 2.3	The FlexCAN signals section is missing. Insert the following section & table: **2.3.11 FlexCAN Signals** Table 23-13 describes the FlexCAN module signals. **Table 23-13 FlexCAN Signals** _(see table below)_
Table 2-18/Page 2-15	Remove sentence "Only present on the MCF5372 and MCF5373 devices." from the Debug Data and Processor Status Outputs entries. Remove "All Processor Status Outputs" entry row because the ALLPST signal is not present.
Table 2-19/Page 2-15	Remove "ALLPST" column in table as well as "(MCF5372 & MCF5373)" in the PST[3:0] column heading.

Table 23-9. LCD Signals

Signal Name	Abbreviation	Function	I/O
Line Data	LCD_D[17:0]	LCD data bus.	O
First Line Marker/ Vertical Sync	LCD_FLM/ LCD_VSYNC	Passive matrix: First line marker Active matrix: Vertical sync pulse. Indicates start of next frame.	O
Line Pulse/ Horizontal Sync	LCD_LP/ LCD_HSYNC	Passive matrix: Line pulse Active matrix: Horizontal sync pulse. Indicates start of next line.	O
Shift Clock	LCD_SCLK	Clock for latching data into the display driver's internal shift register.	O
Alt. Crystal Direction/ Output Enable	LCD_ACD/ LCD_OE	Passive matrix: Alternate crystal direction Active matrix: Output enable to enable data to be shifted onto the display.	O
Contrast	LCD_CONTRAST	Controls the LCD bias voltage for contrast control.	O
Power Save	LCD_PS	Controls signal output for source driver (Sharp HR-TFT 240x320 panels only)	O
Gate Driver Clock Signal	LCD_CLS	Start signal output for gate driver, inverted version of LCD_PS (Sharp HR-TFT 240x320 panels only).	O
Reverse Control	LCD_REV	Signal for common electrode driving signal preparation (Sharp HR-TFT 240x320 panels only).	O
Sampling Start Signal	LCD_SPL_SPR	Sets the horizontal scan direction (Sharp HR-TFT 240x320 panels only).	O

Table 23-13 FlexCAN Signals

Signal Name	Abbreviation	Function	I/O
FlexCAN Transmit	CANTX	Controller area network transmit data output.	O
FlexCAN Receive	CANRX	Controller area network receive data input.	I

Table B-4. MCF5329RM Rev. 0 to Rev. 0.1 Changes (continued)

Location	Description
Section 5.3.1/Page 5-4	Change CACR bit 4 from reserved to EUSP in register figure and bit description table. Add the following for EUSP's bit description: Enable user stack pointer. See Section 3.2.3, "Supervisor/User Stack Pointers (A7 and OTHER_A7)," for more information on the dual stack pointer implementation. 0 Disable the processor's use of the User Stack Pointer 1 Enable the processor's use of the User Stack Pointer
Section 5.4.4/Page 5-12	Change last sentence from: "Therefore, on-chip DMA channels cannot access local memory and do not maintain coherency with the unified cache." to: "Therefore, on-chip DMA channels should not access cached local memory locations, as coherency is not maintained with the unified cache."
Section 5.5/Page 5-16	Remove cache code examples from section, as they are incorrect. Cache code examples can be found within sample projects in the MCF532XSC.ZIP file available at the device web site at http://www.freescale.com/coldfire
Table 6-1/Page 6-2	Remove 1 from RAMBAR register name and mnemonic.
Section 6.2.1/Page 6-2	Add the following two notes: **Note:** By default, the RAMBAR is invalid, but the back door 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, the RAMBAR[V] bit should be set. **Note:** 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 a ghosting effect for the on-chip SRAM memory. For example, writes to addresses 0x8000_0000 and 0x8000_8000 actually modify the exact same memory location. System software should ensure that SRAM address pointers do not exceed the size of the SRAM to prevent unwanted overwriting of SRAM.
Figure 6-1/Page 6-2	Change the reset value for the RAMBAR[BDE] bit from 0 to 1.
Table 11-1/Page 11-2	Correct BCR register address from 0xFC04_002A to 0xFC04_0024.
Figure 11-17/Page 11-10	Correct BCR register address from 0xFC04_002A to 0xFC04_0024.
Chapter 12	Add a reserved master used only for factory test purposes at location M7. The XBS_PRS*n*[M7] field must comply with the restriction that its value must be unique to the other M*n* fields.
Section 12.4.1/Page 12-4	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 (including reserved masters) then valid values are 000 to 110. Unpredictable results occur when using the reserved 111 setting. Update reset values accordingly to 0x6543_0210.
Table 13-1/Page 13-4	Update footnotes in table to correspond to corrections in Table 2-2 as indicated above.
Table 13-14/Page 13-28	In the bit descriptions for the PAR_U0CTS and PAR_U0RTS bits, the description for the bit being set incorrectly refers to UART1 while it should refer to the UART0 port.
Section 17.3.1.1/Page 17-4	Change next to last sentence in section to "For example, a 16-bit address/16-bit data device would connect its addr[15:0] to A[16:1] and data[15:0] to D[31:16]."
Section 17.4.2/Page 17-9	Change the end of the first sentence of the second paragraph from: "...if a longword is transferred for three port sizes when not in split but mode." to "...if a longword is transferred for three port sizes when not in split bus mode."
Figure 19-24/Page 19-23	Correct EMRBR register address from 0xFC03_01B8 to 0xFC03_0188

MCF5329 Reference Manual, Rev 3

Table B-4. MCF5329RM Rev. 0 to Rev. 0.1 Changes (continued)

Location	Description
Section 21.1.3/Page 21-4	Add the following sub-bullet under the "USB device mode" bullet: — Supports full-speed operation via the on-chip transceiver and FS/HS operation using an external ULPI transceiver
Section 21.1.3/Page 21-5	Change the following sub-bullet: "External ULPI transceiver supports high speed (480 Mbps), full speed, and low speed" to — 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.
Table 21-40/Page 21-45	Change "USB Device FS/LS" entry to "USB Device FS". Change "Host/Device ULPI HS/FS/LS" to "Host/Device ULPI HS/FS".
Section 21.4.4.1/Page 21-45	In the note, change the second sentence of the second paragraph from "Device operation requires a 1.5kΩ pull-up resistor on DP (full-speed operation) or DN (low-speed operation) ports." to "Device operation requires a 1.5kΩ pull-up resistor on DP (full-speed operation)." because low-speed operation is not supported in device mode.
Throughout Chapter 23	Remove various *n* and 0 suffixes from register names because there is only one FlexCAN module on the device.
Figure 23-10/Page 23-15	Change IMASK registers address from 0xFC02_002A to 0xFC02_0028.
Section 25.3.6/Page 25-6	Change RTC_ISR[2HZ] bit field name to 2SEC as it more accurately describes the bit definition.
Section 25.3.7/Page 25-7	Change RTC_IER[2HZ] bit field name to 2SEC as it more accurately describes the bit definition.
Section 26.2/Page 26-2	Add a 0x20 offset to all PWM register addresses throughout section. Register addresses should be from 0xFC09_0020 to 0xFC09_0044.
Table 36-21/Page 36-35	Remove 1 from RAMBAR register name and mnemonic.