MCF5301*x* Reference Manual

Devices Supported:

MCF53010	MCF53014
MCF53011	MCF53015
MCF53012	MCF53016
MCF53013	MCF53017

Document Number: MCF53017RM
Rev. 4
8/2009

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

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

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

MCF53017RM
Rev. 4
8/2009

Chapter 1
Overview

Chapter 2
Signal Descriptions

Chapter 3
ColdFire Core

Chapter 4
Enhanced Multiply-Accumulate Unit (EMAC)

Chapter 5
Cache

Chapter 6
Static RAM (SRAM)

Chapter 7
Clock Module

Chapter 8
Power Management

Chapter 9
Chip Configuration Module (CCM)

Chapter 10
Reset Controller Module

Chapter 11
System Control Module (SCM)

Chapter 12
Crossbar Switch (XBS)

Chapter 13
Memory Protection Unit (MPU)

Chapter 14
Pin-Multiplexing and Control

Chapter 15
Interrupt Controller Modules

Chapter 16
Edge Port Modules (EPORT*n*)

Chapter 17
Enhanced Direct Memory Access (eDMA)

Chapter 18
FlexBus

Chapter 19
SDRAM Controller (SDRAMC)

Chapter 20
Universal Serial Bus Interface – Host Module

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

Chapter 22
Enhanced Secure Digital Host Controller

Chapter 23
Cryptographic Acceleration Unit (CAU)

Chapter 24
Random Number Generator (RNG)

Chapter 25
IC Identification (IIM)

Chapter 26
Subscriber Identification Module (SIM)

Chapter 27 Voice Codec

Chapter 28 Fast Ethernet Controllers (FEC0 and FEC1)

Chapter 29
Synchronous Serial Interface (SSI)

Chapter 30
Real-Time Clock

Chapter 31
Programmable Interrupt Timers (PIT0–PIT3)

Chapter 32
DMA Timers (DTIM0–DTIM3)

Chapter 33
DMA Serial Peripheral Interface (DSPI)

Chapter 34
UART Modules

Chapter 35
I^{2}C Interface

Chapter 36
Debug Module

Chapter 37
IEEE 1149.1 Test Access Port (JTAG)

About This Book

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

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

Portions of Chapter 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 this ColdFire processor. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the ColdFire® architecture.

Suggested Reading

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

General Information

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

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

ColdFire Documentation

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

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

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

Conventions

This document uses the following notational conventions:

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

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

|| Field concatenation operator

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

Register Figure Conventions

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

— Undefined at reset.

u Unaffected by reset.

[*signal_name*] Reset value is determined by the polarity of the indicated signal.

The following register fields are used:

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

Chapter 1 Overview

1.1 Introduction

The MCF5301*x* devices are a family of highly-integrated 32-bit microprocessors based on the Version 3 ColdFire microarchitecture. All MCF5301*x* devices contain a smart card interface, an enhanced Secure Digital host controller, USB On-the-Go controllers, a voice-band audio codec, an IC identification module, as well as other peripherals that enable the MCF5301*x* family for use in point-of-sale, voice-over-wirless-LAN, and analog-telephone-adapter applications. Optional peripherals include a USB host controller, second smart card port, cryptography coprocessor, integrated microphone, speaker, handset, and headphone amplifiers, and voice-over-IP software.

This chapter provides an overview of the MCF5301*x* microprocessor family, focusing on its highly diverse feature set. It was written from the perspective of the MCF53017 superset device. However, it also pertains to the MCF53010–16. See the following section for a summary of differences between the various devices of the MCF5301*x* family.

1.2 Features

1.2.1 MCF5301*x* Family Comparison

The following table compares the various device derivatives available within the MCF5301*x* family.

Table 1-1. MCF5301*x* Family Configurations

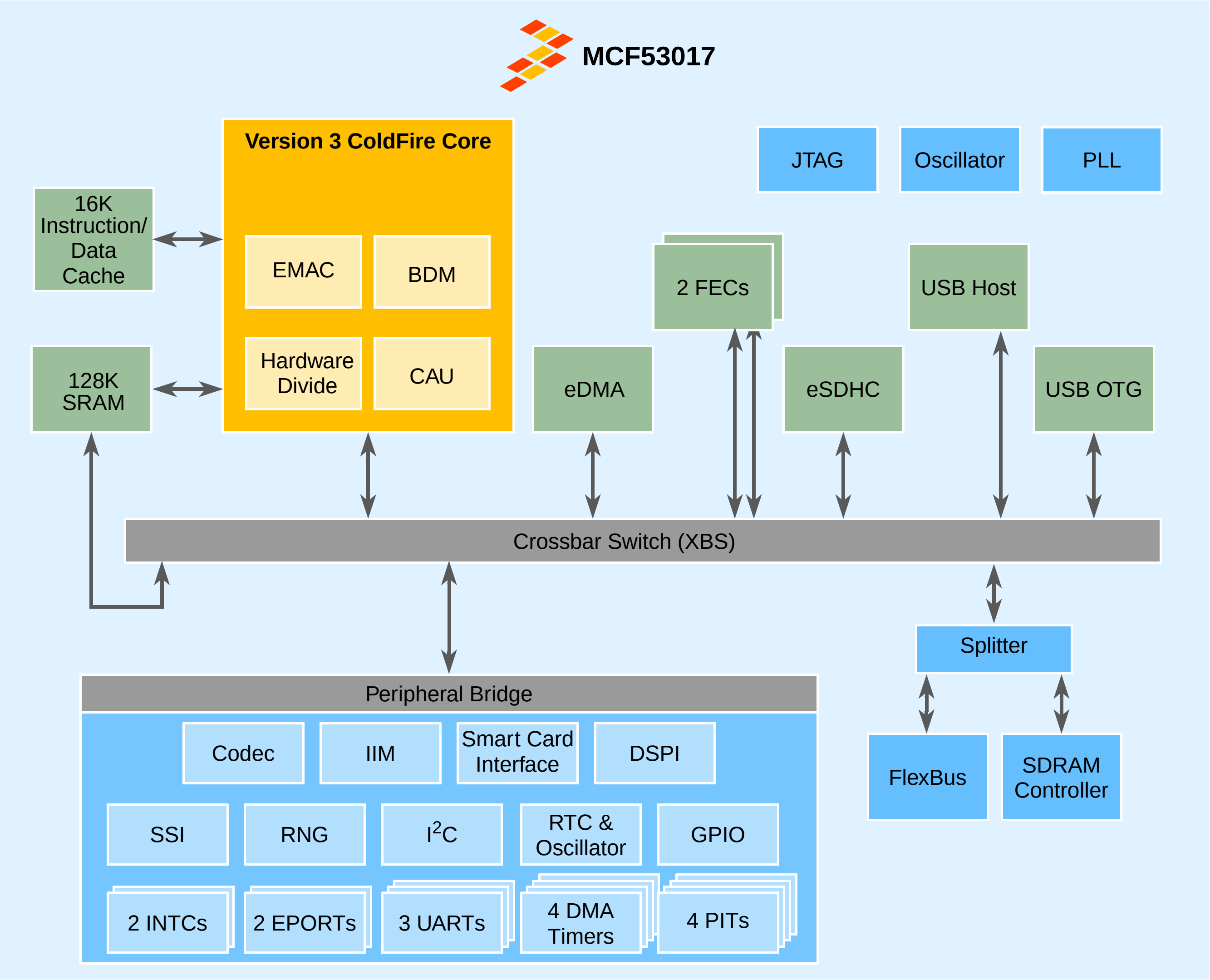
Module	MCF53010	MCF53011	MCF53012	MCF53013	MCF53014	MCF53015	MCF53016	MCF53017
Version 3 ColdFire Core with EMAC (enhanced multiply-accumulate unit)	•	•	•	•	•	•	•	•
Core (system) clock	up to 240 MHz							
Peripheral and external bus clock (Core clock ÷ 3)	up to 80 MHz							
Performance (Dhrystone/2.1 MIPS)	up to 211							
Unified data/instruction cache	16 Kbytes							
Static RAM (SRAM)	128 Kbytes							
Voice-over-IP software	—	—	•	•	—	—	•	•
Cryptography acceleration unit (CAU)	—	•	—	•	—	•	—	•
Random number generator	—	•	—	•	—	•	—	•

Table 1-1. MCF5301*x* Family Configurations (continued)

Module	MCF53010	MCF53011	MCF53012	MCF53013	MCF53014	MCF53015	MCF53016	MCF53017
Smart card interface (SIM)	1 port				2 ports			
Voice-band audio codec	•	•	•	•	•	•	•	•
Integrated audio amplifiers	—	—	—	—	•	•	•	•
IC identification module (IIM)	2 Kbits							
Enhanced Secure Digital host controller (eSDHC)	•	•	•	•	•	•	•	•
SDR/DDR SDRAM controller	•	•	•	•	•	•	•	•
FlexBus external interface	•	•	•	•	•	•	•	•
USB 2.0 On-the-Go	•	•	•	•	•	•	•	•
USB 2.0 Host	—	—	—	—	•	•	•	•
Synchronous serial interface (SSI)	•	•	•	•	•	•	•	•
Fast Ethernet controller (FEC)	2	2	2	2	2	2	2	2
UARTs	3	3	3	3	3	3	3	3
I^{2}C	•	•	•	•	•	•	•	•
DSPI	•	•	•	•	•	•	•	•
Real-time clock	•	•	•	•	•	•	•	•
32-bit DMA timers	4	4	4	4	4	4	4	4
Watchdog timer (WDT)	•	•	•	•	•	•	•	•
Periodic interrupt timers (PIT)	4	4	4	4	4	4	4	4
Edge port module (EPORT)	•	•	•	•	•	•	•	•
Interrupt controllers (INTC)	2	2	2	2	2	2	2	2
16-channel direct memory access (DMA)	•	•	•	•	•	•	•	•
General purpose I/O Module (GPIO)	•	•	•	•	•	•	•	•
JTAG - IEEE® 1149.1 Test Access Port	•	•	•	•	•	•	•	•
Package	208 LQFP				256 MAPBGA			

1.3 Block Diagram

Figure 1-1 shows a top-level block diagram of the MCF53017 superset device.



LEGEND

BDM	– Background debug module	**IIM**	**–** IC identification module
CAU	– Cryptography acceleration unit	**INTC**	**–** Interrupt controller
DSPI	– DMA serial peripheral interface	**JTAG**	– Joint Test Action Group interface
eDMA	– Enhanced direct memory access module	**PCI**	– Peripheral Component Interconnect
eSDHC	– Enhanced Secure Digital host controller	**PIT**	– Programmable interrupt timers
EMAC	– Enchanced multiply-accumulate unit	**PLL**	– Phase locked loop module
EPORT	– Edge port module	**RNG**	– Random number generator
FEC	– Fast Ethernet Controller	**RTC**	– Real time clock
GPIO	– General purpose input/output module	**SSI**	– Synchronous serial interface
I^{2}C	– Inter-Integrated Circuit	**USB OTG**	– Universal Serial Bus On-the-Go controller

Figure 1-1. MCF53017 Block Diagram

1.4 Operating Parameters

- 0ºC to 70ºC junction temperature devices are available
- 1.2V core, 3.3V I/O, 1.8V/2.5V/3.3V external memory bus

1.5 Packages

Depending on device, the MCF5301*x* family is available in the following packages:

- 208-pin low-profile quad flat package (LQFP)
- 256-pin molded array process ball grid array (MAPBGA)

1.6 Chip Level Features

The MCF5301*x* system includes these distinctive features:

- Version 3 ColdFire® core with EMAC
- Up to 211 Dhrystone 2.1 MIPS @ 240 MHz
- 16 Kbytes unified instruction/data cache
- 128 Kbytes internal SRAM
- Crossbar switch technology (XBS) for concurrent access to peripherals or RAM from multiple bus masters
- Enhanced Secure Digital Host Controller (eSDHC)
- Two ISO7816 smart card interfaces
- Voice-band audio codec
- 160 MHz 16-bit (DDR/mobile-DDR) or 80 MHz 32-bit (SDR) SDRAM controller
- USB 2.0 On-the-Go controller
- USB 2.0 host controller
- 2 10/100 Ethernet MACs
- Coprocessor for acceleration of the DES, 3DES, AES, MD5, and SHA-1 algorithms
- Random number generator
- 16 channel DMA controller
- Synchronous serial interface
- 4 periodic interrupt timers
- 4 32-bit timers with DMA support
- DMA supported serial peripheral interface (DSPI)
- 3 UARTs
- I^{2}C bus interface

1.7 Module-by-Module Feature List

The following is a brief summary of the functional blocks in the MCF53017 superset device. For more details refer to the *MCF53017 ColdFire Microprocessor Reference Manual* (MCF53017RM).

1.7.1 Version 3 ColdFire Variable-Length RISC Processor

- Static operation
- 32-bit address and data path on-chip
- Maximum 240 MHz processor core and 80 MHz bus frequency
- Sixteen total general-purpose 32-bit registers data and address
- Enhanced multiply-accumulate unit (EMAC) for DSP and fast multiply operations
- Hardware divide execution unit supporting various 32-bit operations
- Implements the ColdFire Instruction Set Architecture, ISA_A+

1.7.2 Cryptography Acceleration Unit (CAU)

- Instruction-level coprocessor
- Supports DES, 3DES, AES, MD5, and SHA-1

1.7.3 On-chip Memories

- 128 KByte dual-ported SRAM on CPU internal bus
  - Accessible to non-core bus masters (e.g. FEC, DMA, USB OTG, USB host, and eSDHC controllers) via the crossbar switch
- Non-blocking 16 KByte unified cache organized as 4-way set associative with 16 bytes per cache line and 1024 cache lines, supporting copyback and write-through modes of operation

1.7.4 Phase Locked Loop (PLL)

- 14–40 MHz reference crystal
- Four register-programmable output dividers

1.7.5 Power Management

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

1.7.6 Chip Configuration Module (CCM)

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

1.7.7 Reset Controller

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

1.7.8 System Control Module

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

1.7.9 Crossbar Switch

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

1.7.10 Enhanced Secure Digital Host Controller (eSDHC)

- Compatible with the following specifications:
  - — *SD Host Controller Standard Specification, Version 2.0* (http://www.sdcard.org)
  - — *MultiMediaCard System Specification, Version 4.0* (http://www.mmca.org)
  - — *SD Memory Card Specification, Version 2.0* (http://www.sdcard.org)
  - — *SDIO Card Specification, Version 1.2* (http://www.sdcard.org)
  - — *CE-ATA Card Specification, Version 1.0* (http://www.sdcard.org)
- Designed to work with CE-ATA, SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMC plus, MMC 4x, and MMC RS cards
- SD bus clock up to 25 MHz
- Supports 1- and 4-bit SD and SDIO modes, 1- and 4-bit MMC modes, and 4-bit CE-ATA devices
- Contains a fully configurable 128 x 32-bit FIFO for read/write data

1.7.11 Smart Card Interface Module (SIM)

- Two ISO7816 smart card interfaces
- Internal one-wire mode (single data pin)
- Programmable clock divisor
- Fourteen total interrupt sources (six transmit, six receive, two control functions)

1.7.12 Voice Band Audio Codec

- Supports various input clock frequency from 16–30 MHz
- Supports 8 KHz/8.1 KHz software interface frequency
- Includes amplifiers for microphone, speaker, handset, and headphone
- Ability to disable codec (battery save mode) via software
- Ability to disable input/output high-pass filter stage via software
- Programmable decimation filter (250–511.75)

1.7.13 IC Identification Module (IIM)

- 2 Kbits of software-programmable polysilicon fuses
- Eight independent fuse banks of 256 bits each
- Ability to write-protect fuses on a per-bank basis

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

- Support for full speed (FS) and low speed (LS) via a serial interface or on-chip FS/LS transceiver
- Uses 60 MHz reference clock based off of the system clock or from an external pin
- Supports VBUS power enable and VBUS over-current detect to control bus power

1.7.15 Universal Serial Bus (USB) 2.0 Host Controller

- Support for full speed (12 Mbps) and low speed (1.5 Mbps) via serial interface
- On-chip full-speed/low-speed transceiver
- Uses 60 MHz reference clock based off of the system clock or from an external pin
- Supports VBUS power enable and VBUS over-current detect to control bus power

1.7.16 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-byte 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 MByte of memory
- Supports page mode to maximize the data rate
- Supports sleep mode and self-refresh mode

1.7.17 FlexBus (External Interface)

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

1.7.18 Synchronous Serial Interface (SSI)

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

1.7.19 Fast Ethernet Media Access Controllers (FEC MAC)

- Two fast Ethernet controllers
- 10/100 BaseT/TX capability, half duplex or full duplex
- On-chip transmit and receive FIFOs
- Built-in dedicated DMA controller
- Memory-based flexible descriptor rings
- Media independent interface (MII) to external transceiver (PHY)
- Separate RMII gasket to interface with RMII-compatible PHY

1.7.20 Random Number Generator (RNG)

- FIPS-140 compliant for randomness and non-determinism
- Supports key generation
- Integrated entropy sources

1.7.21 Real Time Clock

- Full clock: days, hours, minutes, seconds
- Minute countdown timer with interrupt
- Programmable daily alarm with interrupt
- Sampling timer with interrupt
- Once-per-day, once-per-hour, once-per-minute, and once-per-second interrupts
- User-programmable 16-bit prescaler to support various input clock frequencies

1.7.22 Programmable Interrupt Timers (PIT)

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

1.7.23 DMA Timers

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

1.7.24 DMA Serial Peripheral Interface (DSPI)

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

1.7.25 Universal Asynchronous Receiver Transmitters (UARTs)

- Three UARTs with a 16-bit divider for clock generation
- Interrupt control logic
- DMA support with separate transmit and receive requests
- Programmable clock-rate generator
- Data formats can be 5, 6, 7 or 8 bits with even, odd or no parity
- Up to 2 stop bits in 1/16 increments
- Error-detection capabilities

1.7.26 I^{2}C Module

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

1.7.27 Interrupt Controllers

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

1.7.28 Edge Port Module

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

1.7.29 DMA Controller

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

1.7.30 General Purpose I/O interface

- Up to 83 bits of GPIO for the MCF53012/13 (256 MAPBGA)
- Up to 61 bits of GPIO for the MCF53010/11 (208 LQFP)
- Bit manipulation supported via set/clear functions
- Various unused peripheral pins may be used as GPIO

1.7.31 System Debug Support

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

1.7.32 JTAG Support

- JTAG part identification and part revision numbers

1.8 Memory Map Overview

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

Table 1-2. System Memory Map

Internal Address[31:28]	Address Range	Destination Slave	Slave Memory Size
00xx	0x0000_0000–0x3FFF_FFFF	FlexBus	1024 MB
01xx	0x4000_0000–0x7FFF_FFFF	SDRAM Controller	1024 MB
1000	0x8000_0000–0x8FFF_FFFF	Internal SRAM	256 MB
1001, 101x	0x9000_0000–0xBFFF_FFFF	Reserved	768 MB
110x	0xC000_0000–0xDFFF_FFFF	FlexBus	512 MB
1110	0xE000_0000–0xEFFF_FFFF	Reserved	256 MB
1111	0xF000_0000–0xFFFF_FFFF	Internal Peripheral Space	256 MB

NOTE

This memory map provides two disjoint regions mapped to the FlexBus controller to support glueless connections to external memories (e.g., flash and SRAM), 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 selected since 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] equals 0 identifies the cacheable space.

1.8.1 Internal Peripheral Space

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

Table 1-3. Internal Peripheral Space Memory Map

Base Address	Slot Number	Peripheral
0xFC00_0000	0	SCM (MPR & PACRs)
0xFC00_4000	1	Crossbar switch
0xFC00_8000	2	FlexBus
0xFC01_4000	5	Memory protection unit (MPU)

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

Base Address	Slot Number	Peripheral
0xFC03_0000	12	FEC0
0xFC03_4000	13	FEC1
0xFC04_0000	16	SCM (CWT & core fault registers)
0xFC04_4000	17	eDMA controller
0xFC04_8000	18	Interrupt controller 0
0xFC04_C000	19	Interrupt controller 1
0xFC05_4000	21	Interrupt controller IACK
0xFC05_8000	22	I^2C
0xFC05_C000	23	DSPI
0xFC06_0000	24	UART0
0xFC06_4000	25	UART1
0xFC06_8000	26	UART2
0xFC07_0000	28	DMA timer 0
0xFC07_4000	29	DMA timer 1
0xFC07_8000	30	DMA timer 2
0xFC07_C000	31	DMA timer 3
0xFC08_0000	32	PIT 0
0xFC08_4000	33	PIT 1
0xFC08_8000	34	PIT 2
0xFC08_C000	35	PIT 3
0xFC09_0000	36	Edge port 0
0xFC09_4000	37	Edge port 1
0xFC09_C000	39	Voice codec
0xFC0A_0000	40	CCM, reset controller, power management
0xFC0A_4000	41	Pin multiplexing and control (GPIO)
0xFC0A_8000	42	Real time clock
0xFC0A_C000	43	SIM
0xFC0B_0000	44	USB On-the-Go
0xFC0B_4000	45	USB host
0xFC0B_8000	46	SDRAM controller
0xFC0B_C000	47	SSI
0xFC0C_0000	48	PLL
0xFC0C_4000	49	Random number generator (RNG)

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

Base Address	Slot Number	Peripheral
0xFC0C_8000	50	IIM/Fusebox
0xFC0C_C000	51	eSDHC

1.9 Documentation

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

Chapter 2
Signal Descriptions

2.1 Introduction

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

NOTE

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

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

2.2 Signal Properties Summary

The below table lists the signals grouped by functionality.

NOTE

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., FB_A23), while designations for multiple signals within a group use brackets (i.e., FB_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. Most pins that are muxed with GPIO will default to their GPIO functionality. See Table 2-1 for a list of the exceptions.

Table 2-1. Special-Case Default Signal Functionality

Pin	Default Signal
$\overline{\text{FB\_BE/BWE}}$[3:0]	$\overline{\text{FB\_BE/BWE}}$[3:0]
$\overline{\text{FB\_CS}}$[3:0]	$\overline{\text{FB\_CS}}$[3:0]
$\overline{\text{FB\_OE}}$	$\overline{\text{FB\_OE}}$

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

Pin	Default Signal
$\overline{\text{FB\_TA}}$	$\overline{\text{FB\_TA}}$
FB_R/$\overline{\text{W}}$	FB_R/$\overline{\text{W}}$
$\overline{\text{FB\_TS}}$	$\overline{\text{FB\_TS}}$

Table 2-2. MCF5301*x* Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
Reset								
$\overline{\text{RESET}}$	—	—	—	U	I	EVDD	41	M3
$\overline{\text{RSTOUT}}$	—	—	—	—	O	EVDD	42	N1
Clock								
EXTAL	—	—	—	—	I	EVDD	49	T2
XTAL	—	—	—	U[3]	O	EVDD	50	T3
Mode Selection								
BOOTMOD[1:0]	—	—	—	—	I	EVDD	55, 17	J5, G5
FlexBus								
FB_A[23:22]	—	$\overline{\text{FB\_CS}}$[3:2]	—	—	O	SDVDD	115, 114	P16, N16
FB_A[21:16]	—	—	—	—	O	SDVDD	113–108	R16, N14, N15, P15-13
FB_A[15:14]	—	SD_BA[1:0]	—	—	O	SDVDD	107, 106	R15, R14
FB_A[13:11]	—	SD_A[13:11]	—	—	O	SDVDD	105–103	N13, R12, R13
FB_A10	—	—	—	—	O	SDVDD	100	N12
FB_A[9:0]	—	SD_A[9:0]	—	—	O	SDVDD	99–97 95–89	P12, T14, T15, R11, P11, N11, T13, R10, T11, T12
FB_D[31:16]	—	SD_D[31:16]	—	—	I/O	SDVDD	208–198, 57–62, 64, 65	B3, A2, D6, C5, B4, A3, B5, C6, D12, C14, B14, C13, D11, B13, A14, A13
FB_D[15:0]	—	FB_D[31:16]	—	—	I/O	SDVDD	182–189, 177–170	B9, A9, A8, D7, B8, C8, D8, B7, C10, A10, B10, D10, C11, A11, B11, A12
FB_CLK	—	—	—	—	O	SDVDD	153	D13
$\overline{\text{FB\_BE/BWE}}$[3:0]	PBE[3:0]	SD_DQM[3:0]	—	—	O	SDVDD	197, 166, 179, 178	A4, B12, C9, D9
$\overline{\text{FB\_CS}}$[5:4]	PCS[5:4]	—	—	—	O	SDVDD	—	B6, C7
$\overline{\text{FB\_CS1}}$	PCS1	$\overline{\text{SD\_CS1}}$	—	—	O	SDVDD	5	D2

Table 2-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
FB_CS0	PCS0	FB_CS4	—	—	O	SDVDD	6	C2
FB_OE	PFBCTL3	—	—	—	O	SDVDD	1	D4
FB_TA	PFBCTL2	—	—	U	I	SDVDD	3	B2
FB_R/W	PFBCTL1	—	—	—	O	SDVDD	2	C3
FB_TS	PFBCTL0	DACK0	—	—	O	SDVDD	4	D3
SDRAM Controller								
SD_A10	—	—	—	—	O	SDVDD	206	C4
SD_CAS	—	—	—	—	O	SDVDD	154	D15
SD_CKE	—	—	—	—	O	SDVDD	151	B15
SD_CLK	—	—	—	—	O	SDVDD	190	A7
SD_CLK	—	—	—	—	O	SDVDD	191	A6
SD_CS0	—	—	—	—	O	SDVDD	155	A15
SD_DQS[1:0]	—	—	—	—	O	SDVDD	196, 167	C12, A5
SD_RAS	—	—	—	—	O	SDVDD	152	C15
SD_SDR_DQS	—	—	—	—	I	SDVDD	207	D5
SD_WE	—	—	—	—	O	SDVDD	150	D14
External Interrupts Port 1[4,5]								
IRQ1DEBUG[7:4]	PIRQ1DEBUG [7:4]	DDATA[3:0]	—	—	I	EVDD	—	H1, H4-2
IRQ1DEBUG[3:0]	PIRQ1DEBUG [3:0]	PST[3:0]	—	—	I	EVDD	—	K14, H14, K15, J13
IRQ1FEC7	PIRQ1FEC7	RMII1_CRS_DV	MII0_CRS	—	I	EVDD	29	J1
IRQ1FEC6	PIRQ1FEC6	RMII1_RXER	MII0_RXCLK	—	I	EVDD	30	J2
IRQ1FEC5	PIRQ1FEC5	RMII1_TXEN	MII0_TXCLK	—	I	EVDD	31	K4
IRQ1FEC4	PIRQ1FEC4	RMII1_REF_CLK	—	D	I	EVDD	32	J3
IRQ1FEC[3:2]	PIRQ1FEC[3:2]	RMII1_RXD[1:0]	MII0_RXD[3:2]	—	I	EVDD	33, 34	J4, K1
IRQ1FEC[1:0]	PIRQ1FEC[1:0]	RMII1_TXD[1:0]	MII0_TXD[3:2]	—	I	EVDD	35, 36	K2, L1
External Interrupts Port 0[5]								
IRQ07	PIRQ07	—	—	U	I	EVDD	10	E4
IRQ06	PIRQ06	—	USB_CLKIN	U	I	EVDD	—	L13

Table 2-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
$\overline{\text{IRQ04}}$	PIRQ04	$\overline{\text{DREQ0}}$	—	U	I	EVDD	19	D1
$\overline{\text{IRQ01}}$	PIRQ01	$\overline{\text{DREQ1}}$	—	U	I	EVDD	11	F4
Enhanced Secure Digital Host Controller								
SDHC_DAT3	PSDHC5	—	—	UD	I/O	EVDD	60	N4
SDHC_DAT[2:0]	PSDHC[4:2]	—	—	U	I/O	EVDD	61–63	R5, N6, N5
SDHC_CMD	PSDHC1	—	—	U	I/O	EVDD	59	R4
SDHC_CLK	PSDHC0	—	—	—	O	EVDD	58	R3
Codec								
CODEC_ADCN	—	AMP_MICN	—	—	I		85	P10
CODEC_ADCP	—	AMP_MICP	—	—	I		84	P9
CODEC_BGRVREF	—	—	—	—	I		86	N9
CODEC_DACN	—	AMP_HSN	—	—	O		75	R7
CODEC_DACP	—	AMP_HSP	—	—	O		67	R6
CODEC_REGBYP	—	—	—	—	I		81	P6
CODEC_REFN	—	—	—	—	I		79	P8
CODEC_REFP	—	—	—	—	I		78	P7
CODEC_VAG	—	—	—	—	I		82	N7
Amplifiers								
AMP_HPDUMMY	—	—	—	—	O		—	R9
AMP_HPOUT	—	—	—	—	O		—	R8
AMP_SPKRN	—	—	—	—	O		—	T9
AMP_SPKRP	—	—	—	—	O		—	T7
Smart Card interface 1								
SIM1_DATA	PSIM14	SSI_TXD	U1TXD	UD	I/O	EVDD	141	E14
SIM1_VEN	PSIM13	SSI_RXD	U1RXD	UD	O	EVDD	142	D16
SIM1_RST	PSIM12	SSI_FS	$\overline{\text{U1RTS}}$	—	O	EVDD	144	E13
SIM1_PD	PSIM11	SSI_BCLK	$\overline{\text{U1CTS}}$	—	O	EVDD	145	E15
SIM1_CLK	PSIM10	SSI_MCLK	—	—	O	EVDD	143	F13

Table 2-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
Smart Card interface 0								
SIM0_DATA	PSIM04	—	—	—	I/O	EVDD	—	L3
SIM0_VEN	PSIM03	—	—	—	O	EVDD	—	M2
SIM0_RST	PSIM02	—	—	—	O	EVDD	—	F16
SIM0_PD	PSIM01	—	—	—	O	EVDD	—	L14
SIM0_CLK	PSIM00	—	—	—	O	EVDD	—	M16
USB On-the-Go								
USBO_DM	—	—	—	—	O	USB VDD	148	C16
USBO_DP	—	—	—	—	O	USB VDD	149	B16
USB Host								
USBH_DM	—	—	—	—	O	USB VDD	—	B1
USBH_DP	—	—	—	—	O	USB VDD	—	C1
FEC 1								
RMII1_MDC	PFECI2C5	—	MII0_TXER	—		EVDD	22	E1
RMII1_MDIO	PFECI2C4	—	MII0_COL	—		EVDD	23	F1
FEC 0								
RMII0_CRS_DV	PFEC06	—	MII0_RXDV	—		EVDD	131	G16
RMII0_RXD[1:0]	PFEC0[5:4]	—	MII0_RXD[1:0]	—		EVDD	130, 129	H15, H16
RMII0_RXER	PFEC03	—	MII0_RXER	—		EVDD	127	J16
RMII0_TXD[1:0]	PFEC0[2:1]	—	MII0_TXD[1:0]	—		EVDD	125, 124	J15, J14
RMII0_TXEN	PFEC00	—	MII0_TXEN	D		EVDD	123	K16
RMII0_MDC	PFECI2C3	—	MII0_MDC	—		EVDD	133	G14
RMII0_MDIO	PFECI2C2	—	MII0_MDIO	—		EVDD	132	G15
Real Time Clock								
RTC_EXTAL	—	—	—	—	I	EVDD	—	P1
RTC_XTAL	—	—	—	—	O	EVDD	—	R1

Table 2-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
Synchronous Serial Interface								
SSI_RXD	PSSI4	—	U1RXD	UD	I	EVDD	—	N3
SSI_TXD	PSSI3	—	U1TXD	UD	O	EVDD	—	P3
SSI_FS	PSSI2	—	$\overline{\text{U1RTS}}$	—	I/O	EVDD	—	R2
SSI_MCLK	PSSI1	—	SSI_CLKIN	—	O	EVDD	—	P4
SSI_BCLK	PSSI0	—	$\overline{\text{U1CTS}}$	—	I/O	EVDD	—	P5
I^{2}C								
I2C_SCL	PFECI2C1	U2RXD	RMII1_MDC	U	I/O	EVDD	37	M1
I2C_SDA	PFECI2C0	U2TXD	RMII1_MDIO	U	I/O	EVDD	38	K3
DSPI								
DSPI_PCS3	PDSPI6	USBH_VBUS_EN	—	—	I/O	EVDD	—	P2
DSPI_PCS2	PDSPI5	USBH_VBUS_OC	—	—	I/O	EVDD	—	N2
DSPI_PCS1	PDSPI4	—	—	—	I/O	EVDD	140	F14
DSPI_PCS0/$\overline{\text{SS}}$	PDSPI3	$\overline{\text{U2RTS}}$	—	U	I/O	EVDD	137	G13
DSPI_SCK	PDSPI2	$\overline{\text{U2CTS}}$	—	—	I/O	EVDD	134	H13
DSPI_SIN	PDSPI1	U2RXD	—	—	I	EVDD	136	E16
DSPI_SOUT	PDSPI0	U2TXD	—	—	O	EVDD	135	F15
UARTs								
U2RXD	PUART5	—	—	—	I	EVDD	14	E2
U2TXD	PUART4	—	—	—	O	EVDD	18	F2
$\overline{\text{U0CTS}}$	PUART3	USBO_VBUS_EN	USB_PULLUP	—	I	EVDD	20	G4
$\overline{\text{U0RTS}}$	PUART2	USBO_VBUS_OC	—	—	O	EVDD	21	G3
U0RXD	PUART1	—	—	—	I	EVDD	27	G2
U0TXD	PUART0	—	—	—	O	EVDD	28	G1
DMA Timers								
T3IN	PTIMER3	T3OUT	IRQ03	—	I	EVDD	13	F3
T2IN	PTIMER2	T2OUT	IRQ02	—	I	EVDD	12	E3
T1IN	PTIMER1	T1OUT	$\overline{\text{DACK1}}$	—	I	EVDD	122	K13
T0IN	PTIMER0	T0OUT	CODEC_ALTCLK	—	I	EVDD	121	L16

Table 2-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
BDM/JTAG[6]								
ALLPST	PDEBUG	—	—	—	O	EVDD	43	—
JTAG_EN	—	—	—	D	I	EVDD	64	M8
PSTCLK	—	TCLK	—	—	I	EVDD	65	T5
DSI	—	TDI	—	U	I	EVDD	66	T4
DSO	—	TDO	—	—	O	EVDD	120	M15
$\overline{\text{BKPT}}$	—	TMS	—	U	I	EVDD	119	M14
DSCLK	—	$\overline{\text{TRST}}$	—	U	I	EVDD	118	L15
Test								
TEST	—	—	—	D	I	EVDD	146	F12
Power Supplies								
IVDD	—	—	—	—	—	—	16, 44, 69, 77, 128, 169, 193	E9, F8, F9, H5, H6, H11, H12, J6, J11, L8, L9
EVDD	—	—	—	—	—	—	9, 24, 26, 40, 47, 51, 54, 57, 74, 126, 139, 195	F5, G6, G11, G12, J12, K6, K11, K12, L5-7, L10-12, M5-7, M12
SD_VDD	—	—	—	—	—	—	7, 102, 116, 156, 163, 181, 208	E5, E6, E10-12, F6, F7, F10, F11
VDD_OSC_A_PLL	—	—	—	—	—	—	46	M4
VDD_USBO	—	—	—	—	—	—	147	E7
VDD_USBH	—	—	—	—	—	—	—	E8
VDD_RTC	—	—	—	—	—	—	—	
AVDD_CODEC	—	—	—	—	—	—	80	N8
AVDD_SPKR	—	—	—	—	—	—	—	T8
VDD_EPM	—	—	—	—	—	—	96	M9
VSTBY_SRAM	—	—	—	—	—	—	—	L2
VSTBY_RTC	—	—	—	—	—	—	—	L4
VSS	—	—	—	—	—	—	8, 15, 25, 39, 45, 48, 52, 53, 56, 68, 73, 76, 101, 117, 138, 168, 180, 192, 194	A1, A16, G7-10, H7-10, J7-10, K7-10, T1, T16
VSS_CODEC	—	—	—	—	—	—	83	N10

Table 2-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
AVSS_SPKR_HDST	—	—	—	—	—	—	—	T6
AVSS_SPKR_HP	—	—	—	—	—	—	—	T10

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

[2] Refers to pin’s primary function.

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

[4] The edge port 1 signals are the primary functions on two sets of pins (IRQ1FEC*n* and IRQ1DEBUG*n*). If an IRQ1 function is configured on both pins, the IRQ1FEC*n* pin takes priority. The corresponding IRQ1DEBUG*n* pin is disconnected internally from the edge port 1 module.

[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.

2.3 Signal Primary Functions

2.3.1 Reset Signals

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

Table 2-3. Reset Signals

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

2.3.2 PLL and Clock Signals

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

Table 2-4. PLL and Clock Signals

Signal Name	Abbreviation	Function	I/O
External clock in	EXTAL	Always driven by an external clock input except when used as a connection to the external crystal if the internal oscillator circuit is used. Clock source may be configured during reset. See Chapter 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
RTC external clock in	RTC_EXTAL	Crystal input clock for the real-time clock module.	I
RTC crystal	RTC_XTAL	Oscillator output to EXTAL RTC crystal.	O
FlexBus clock out	FB_CLK	Reflects the internal bus clock (or one-half the core/system clock). ($f_{sys/2}$)	O
USB clock in	USB_CLKIN	This pin allows the user to drive the reference clock to the USB module as an alternate method of generating the USB reference clock during FS/LS operation. This pin should be driven only with a 60 MHz clock.	I
SSI clock in	SSI_CLKIN	This pin allows the user to drive a specific clock frequency to the SSI module.	I
Codec clock in	CODEC_ALTCLK	Optional external clock signal to the codec. The voice codec supports seven clock frequencies: 20 MHz, 24 MHz, 26 MHz, 28 MHz, 30 MHz, 19.44 MHz and 16.8 MHz.	I

2.3.3 Mode Selection

Table 2-5. Mode Selection Signals

Signal Name	Abbreviation	Function	I/O
Boot mode	BOOTMOD[1:0]	Selects the device's boot mode and chip configuration at reset. See Chapter 9, "Chip Configuration Module (CCM)," for the signal encodings.	I

2.3.4 Enhanced Secure Digital Host Controller

Table 2-6. eSDHC Signals

Signal Name	Abbreviation	Function	I/O
SDHC data bus	SDHC_DAT[3:0]	Data lines. SDHC_DAT3 can be used as card-detection input.	I/O
SDHC command	SDHC_CMD	Command line	I/O
SDHC clock	SDHC_CLK	Clock for MMC/SD/SDIO card	O

2.3.5 SmartCard Interface Ports

Table 2-7. SIM Port Signals

Signal Name	Abbreviation	Function	I/O
SIM data	SIM*n*_DATA	Bidirectional transmit/receive data signal	I/O
SIM supply enable	SIM*n*_VEN	Power supply enable signal	O
SIM reset	SIM*n*_RST	Reset signal	O
SIM card detection	SIM*n*_PD	Card insertion detect signal	O
SIM clock	SIM*n*_CLK	Clock for the smart card. Typical frequencies are 1–5 MHz. This clock is 372 times the data rate that is on SIM_DATA. There is no required timing relationship between this clock signal and any of the other data signals. This is because of the asynchronous nature of the protocol.	O

2.3.6 Voice Codec and Amplifiers

Table 2-8. Voice Codec and Amplifier Signals

Signal Name	Abbreviation	Function	I/O
A/D input	CODEC_ADCN, CODEC_ADCP	Differential analog input to the A/D of the voice codec. By default, these signals are driven by the internal microphone amplifier. If the microphone amplifier is bypassed, they can be driven externally.	I
D/A output	CODEC_DACN, CODEC_DACP	Differential analog outputs of the D/A of the voice codec. By default, the DAC outputs drive the speaker, handset, and headphone audio driver amplifiers. If the audio amplifiers are bypassed, the DAC outputs are driven externally and external amplifiers can be used.	O
Bandgap output voltage	CODEC_BGRVREF	Bandgap output voltage. Connect with a 0.1 μF bypass capacitor to VSS_CODEC.	I
Codec regulator output	CODEC_REGBYP	Output of codec regulator to provide power supply to the ADC, DAC, REF, and microphone amplifier blocks. Connect with a 1 μF external capacitor to VSS_CODEC.	I
ADC reference voltage	CODEC_REFN, CODEC_REFP	Differential reference voltage signals for the analog part of the ADC. Connect each with 0.1 a μF bypass capacitor to VSS_CODEC. **Note:** It is important to connect these capacitors to the same point on the VSS_CODEC trace.	I
Common mode reference	CODEC_VAG	Analog common mode reference for analog ADC and DAC part of voice codec and audio amplifiers. Connect with a 0.1 μF bypass capacitor to VSS_CODEC.	I
Microphone input	AMP_MICN, AMP_MICP	Differential analoginput signals of the microphone amplifier.	I

Table 2-8. Voice Codec and Amplifier Signals (continued)

Signal Name	Abbreviation	Function	I/O
Headset output	AMP_HDSTN, AMP_HDSTP	Differential analog output signals of the headset amplifier.	O
Headphone dummy load	AMP_HPDUMMY	AC coupling of the dummy load of the headphone amplifier to GND. The dummy load is needed for stability of headphone amp for high-capacitance, low-conductance loads. The capacitor to GND at AMP_HPDUMMY output is also used for slow ramp-up of common mode voltage for headphone amplifier.	O
Headphone output	AMP_HPOUT	Single-ended analog output. An onboard AC coupling cap should be connected between AMP_HPOUT and the load.	O
Spekaer output	AMP_SPKRN, AMP_SPKRP	Differential analog output signals of the speaker amplifier.	O

2.3.7 FlexBus Signals

Table 2-9 describes signals that are used for performing transactions on the external bus.

Table 2-9. FlexBus Signals

Signal Name	Abbreviation	Function	I/O
Address bus	FB_A[23:0]	Defines 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	FB_D[31:0]	Provides the general purpose data path between the processor and attached devices.	I/O
Byte enables	FB_BE/BWE[3:0]	Defines flow of data on data bus. During peripheral accesses, these output signals indicate that data is to be latched or driven onto a byte of the data bus when driven low. The BE/BWE[3:0] signals are asserted only to the memory bytes used during a read or write access. BE/BWE0 controls access to the most significant byte lane of data, and BE/BWE3 controls access to the least significant byte lane of data. For SRAM or Flash devices, the BE/BWE*n* outputs should be connected to individual byte strobe signals. The BE/BWE*n* signals are asserted during accesses to on-chip peripherals, but not to on-chip SRAM or cache.	O
Output enable	FB_OE	Indicates when an external device can drive data during external read cycles.	O
Transfer acknowledge	FB_TA	Indicates external data transfer is complete. During a read cycle, when the processor recognizes TA, it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes TA, the bus cycle is terminated.	I

Table 2-9. FlexBus Signals (continued)

Signal Name	Abbreviation	Function	I/O
Read/Write	FB_R/$\overline{\text{W}}$	Indicates direction of the data transfer on the bus for SRAM (R/$\overline{\text{W}}$) 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	$\overline{\text{FB\_TS}}$	Bus control output signal indicating the start of a transfer.	O
Chip selects	$\overline{\text{FB\_CS}}$[5:0]	Select external devices for external bus transactions.	O

2.3.8 SDRAM Controller Signals

Table 2-10 describes signals used for SDRAM accesses.

Table 2-10. SDRAM Controller Signals

Signal Name	Abbreviation	Function	I/O
SDRAM address bus	SD_A[13:0]	Address bus used for multiplexed row and column addresses during SDRAM bus cycles.	O
SDRAM data bus	SD_D[31:16]	Bidirectional, non-multiplexed data bus for SDRAM accesses.	I/O
SDRAM bank address	SD_BA[1:0]	Selects one of the two SDRAM row banks.	O
SDRAM clock enable	SD_CKE	SDRAM clock enable.	O
DDR SDRAM clock	SD_CLK	Output clock for DDR SDRAM.	O
DDR SDRAM clock	$\overline{\text{SD\_CLK}}$	Inverted output clock for DDR SDRAM.	O
SDRAM chip selects	$\overline{\text{SD\_CS}}$[1:0]	SDRAM chip select signals.	O
DDR SDRAM data strobes	SD_DQS[1:0]	Indicates when valid data is on data bus.	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 the SD_DQS inputs.	O
SDRAM write data byte mask	SD_DQM[3:0]	Determines which byte lanes of data bus should be latched during a write cycle. The SD_DQM*n* should be connected to individual SDRAM DQM signals. Most SDRAMs associate DQM3 with the MSB, in which case SD_DQM3 should be connected to the SDRAM's DQM3 input.	O
SDRAM column address strobe	$\overline{\text{SD\_CAS}}$	SDRAM column address strobe.	O
SDRAM row address strobe	$\overline{\text{SD\_RAS}}$	SDRAM row address strobe.	O
SDRAM write enable	$\overline{\text{SD\_WE}}$	Indicates direction of data transfer on bus for SDRAM accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device.	O

2.3.9 External Interrupt Signals

Table 2-11. External Interrupt Signals

Signal Name	Abbreviation	Function	I/O
Edge port 1	$\overline{\text{IRQ1DEBUG}}$[7:0]/ $\overline{\text{IRQ1FEC}}$[7:0]	External interrupt sources. **Note:** The edge port 1 signals are available on either the FEC1 or debug ports. If both sets of pins are configured as edge port 1, the IRQ1FEC*n* pins function as edge port signals and the IRQ1DEBUG*n* pins are disconnected from the edge port module.	I
Edge port 0	$\overline{\text{IRQ0}}$[7,6,4:1]	External interrupt sources.	I

2.3.10 DMA Signals

Table 2-12. DMA Signals

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

2.3.11 Fast Ethernet Controller (FEC0 and FEC1) Signals

The processor contains two Ethernet controllers. However, due to external pin limitations, if MII mode is used, only FEC0 can be used. In RMII mode, both Ethernet controllers are available.

Table 2-13. FEC Signal Descriptions

Signal Name	MII	7-wire	RMII	Description
FEC*n*_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*n*_CRS	X	—	X	Carrier sense. When asserted, indicates transmit or receive medium is not idle. **Note:** In RMII mode, this pin also generates the DV signal.
FEC*n*_MDC	X	—	—	Output clock provides a timing reference to the PHY for data transfers on the FEC*n*_MDIO signal.
FEC*n*_MDIO	X	—	—	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC*n*_MDC. This signal is an input after reset. When the FEC operates in 10Mbps 7-wire interface mode, this signal should be connected to VSS.
FEC*n*_RXCLK	X	X	—	Provides a timing reference for FEC*n*_RXDV, FEC*n*_RXD[3:0], and FEC*n*_RXER.
FEC*n*_RXDV	X	X	—	Asserting the FEC*n*_RXDV input indicates PHY has valid nibbles present on the MII. FEC*n*_RXDV must remain asserted from the first recovered nibble of the frame through to the last nibble. Assertion of FEC*n*_RXDV must start no later than the SFD and exclude any EOF. **Note:** In RMII mode, this signal is present on the FEC*n*_CRS pin.
FEC*n*_RXD0	X	X	X	This pin contains the Ethernet input data transferred from PHY to the media-access controller when FEC*n*_RXDV is asserted.
FEC*n*_RXD1	X	—	X	This pin contains the Ethernet input data transferred from PHY to the media access controller when FEC*n*_RXDV is asserted.

Table 2-13. FEC Signal Descriptions (continued)

Signal Name	MII	7-wire	RMII	Description
FEC*n*_RXD[3:2]	X	—	—	These pins contain the Ethernet input data transferred from PHY to the media access controller when FEC*n*_RXDV is asserted.
FEC*n*_RXER	X	—	X	When asserted with FEC*n*_RXDV, indicates PHY detects an error in the current frame. When FEC*n*_RXDV is not asserted, FEC*n*_RXER has no effect.
FEC*n*_TXCLK	X	X	X	Input clock which provides a timing reference for FEC*n*_TXEN, FEC*n*_TXD[3:0] and FEC*n*_TXER. **Note:** In RMII mode, this signal is the reference clock for receive, transmit, and the control interface.
FEC*n*_TXD0	X	X	X	The serial output Ethernet data and only valid during the assertion of FEC*n*_TXEN.
FEC*n*_TXD1	X	—	X	This pin contains the serial output Ethernet data and valid only during assertion of FEC*n*_TXEN.
FEC*n*_TXD[3:2]	X	—	—	These pins contain the serial output Ethernet data and valid only during assertion of FEC*n*_TXEN.
FEC*n*_TXEN	X	X	X	Indicates when valid nibbles are present on the MII. This signal is asserted with the first nibble of a preamble and is negated before the first FEC*n*_TXCLK following the final nibble of the frame.
FEC*n*_TXER	X	—	—	When asserted for one or more clock cycles while FEC*n*_TXEN is also asserted, PHY sends one or more illegal symbols. FEC*n*_TXER has no effect at 10 Mbps or when FEC*n*_TXEN is negated.

2.3.12 I^{2}C I/O Signals

Table 2-14. I^{2}C I/O Signals

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

2.3.13 DMA Serial Peripheral Interface (DSPI) Signals

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

Signal Name	Abbreviation	Function	I/O
DSPI synchronous serial output	DSPI_SOUT	Provides the serial data from the DSPI and can be programmed to be driven on the rising or falling edge of DSPI_SCK. Each byte is sent msb first.	O
DSPI synchronous serial data input	DSPI_SIN	Provides the serial data to the DSPI and can be programmed to be sampled on the rising or falling edge of DSPI_SCK. Each byte is written to RAM lsb first.	I
DSPI serial clock	DSPI_SCK	Provides the serial clock from the DSPI. In master mode, the processor generates DSPI_SCK, while in slave mode, DSPI_SCK is an input from an external bus master.	I/O
DSPI peripheral chip selects	DSPI_PCS[3:1]	Provide DSPI peripheral chip selects that can be programmed to be active high or low.	O
DSPI peripheral chip select 0/slave select	DSPI_PCS0/ $\overline{\text{DSPI\_SS}}$	In master mode, DSPI_PCS0 is a peripheral chip select output that selects which slave device the current transmission is intended. In slave mode, the SS signal is a slave select input that allows an SPI master to select the processor as the target for transmission.	I/O

2.3.14 Synchronous Serial Interface (SSI) Signals

Table 2-16. SSI Module Signals

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

2.3.15 Universal Serial Bus (USB) Signals

Table 2-17. USB Module Signals

Signal Name	Abbreviation	Function	I/O
USB On-the-Go D-	USBO_DM	D- output of the dual-speed transceiver for the On-the-Go module.	O
USB On-the-Go D+	USBO_DP	D+ output of the dual-speed transceiver for the On-the-Go module.	O

Table 2-17. USB Module Signals (continued)

Signal Name	Abbreviation	Function	I/O
USB OTG VBUS enable	USBO_VBUS_EN	Enables the off-chip VBUS charge pump when USB OTG module is configured as a host.	O
USB OTG VBUS over-current	USBO_VBUS_OC	Indicates to the processor that a short has occurred on USB data bus.	I
USB host D-	USBH_DM	D- output of the dual-speed transceiver for the host module.	O
USB host D+	USBH_DP	D+ output of the dual-speed transceiver for the host module.	O
USB host VBUS enable	USBH_VBUS_EN	Enables the off-chip VBUS charge pump when USB host module is configured as a host.	O
USB host VBUS over-current	USBH_VBUS_OC	Indicates to the processor that a short has occurred on USB data bus.	I
USB external pullup Enable	USB_PULLUP	Either use this pullup enable output signal, or turn it off in the CCM's MISCCR[USBPUE] bit. If internal pullup (and not this output signal) is used, the internal pullup automatically switches impedances based on whether USB is transmitting or receiving.	O

2.3.16 UART Module Signals

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

Table 2-18. UART Module Signals

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

2.3.17 DMA Timer Signals

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

Table 2-19. DMA Timer Signals

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

2.3.18 Debug Support Signals

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

Table 2-20. Debug Support Signals

Signal Name	Abbreviation	Function	I/O
JTAG enable	JTAG_EN	Enables JTAG (asserted) or BDM (negated) operation.	I
JTAG Signals			
Test reset	$\overline{\text{TRST}}$	Active-low signal used to initialize the JTAG logic asynchronously.	I
Test clock	TCLK	Used to synchronize the JTAG logic.	I
Test mode select	TMS	Used to sequence the JTAG state machine. TMS is sampled on the rising edge of TCLK.	I
Test data input	TDI	Serial input for test instructions and data. TDI is sampled on the rising edge of TCLK.	I
Test data output	TDO	Serial output for test instructions and data. TDO is three-stateable and actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O
BDM Signals			
Development serial clock	DSCLK	Clocks the serial communication port to the BDM module during packet transfers.	I
Breakpoint	$\overline{\text{BKPT}}$	Used to request a manual breakpoint.	I
Development serial input	DSI	Internally-synchronized signal provides data input for the serial communication port to the BDM module.	I
Development serial output	DSO	Internally-registered signal provides serial output communication for BDM module responses.	O

Table 2-20. Debug Support Signals (continued)

Signal Name	Abbreviation	Function	I/O
Processor status clock	PSTCLK	Used by the development system to know when to sample 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]. **Note:** Only present on the BGA devices.	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]. **Note:** Only present on the BGA devices.	O
All processor status outputs	ALLPST	ALLPST is a logical AND of the four PST signals and is present in place of PST[3:0] and DDATA[3:0] on the QFP devices. When asserted, reflects that the core is halted.	O

Table 2-21. Processor Status

PST[3:0] (BGA Devices)	ALLPST (QFP Devices)	Processor Status
0000	0	Continue execution
0001	0	Begin execution of one instruction
0010	0	Reserved
0011	0	Entry into user mode
0100	0	Begin execution of PULSE and WDDATA instructions
0101	0	Begin execution of taken branch
0110	0	Reserved
0111	0	Begin execution of RTE instruction
1000	0	Begin one-byte transfer on PSTDDATA
1001	0	Begin two-byte transfer on PSTDDATA
1010	0	Begin three-byte transfer on PSTDDATA
1011	0	Begin four-byte transfer on PSTDDATA
1100	0	Exception processing
1101	0	Reserved
1110	0	Processor is stopped
1111	1	Processor is halted

2.3.19 Test Signals

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

Table 2-22. Test Signals

Signal Name	Abbreviation	Function	I/O
Test	TEST	Reserved for factory testing only and in normal modes of operation should be connected to VSS to prevent unintentional activation of test functions.	I

2.3.20 Power and Ground Pins

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

Table 2-23. Power and Ground Pins

Signal Name	Abbreviation	Function	I/O
PLL analog supply	VDD_A_PLL	Dedicated power supply signal to isolate the sensitive PLL analog (VCO) circuitry from the normal levels of noise present on the digital power supply.	—
Oscillator	VDD_OSC VSS_OSC	Dedicated power supply signals to isolate the sensitive oscillator circuitry from the normal levels of noise present on the digital power supply.	—
Positive I/O supply	EVDD	These pins supply positive power to the I/O pads.	—
Positive core supply	IVDD	These pins supply positive power to the core logic.	—
SDRAMC supply	SD_VDD	These pins supply positive power to the SDRAM controller.	—
USB On-the-Go supply	VDD_USBO	This pin supplies positive power to the USB OTG controller.	—
USB host supply	VDD_USBH	This pin supplies positive power to the USB host controller.	—
Real-time clock supply	VDD_RTC	These pins supply positive power to the RTC module.	—
Ground	VSS	These pins are the negative supply (ground) for the device.	—
Codec supply	AVDD_CODEC VSS_CODEC	Analog supply voltage for the codec regulator and the bandgap reference blocks.	—
Codec amplifiers supply	AVDD_SPKR VSS_SPKR_HDST AVSS_SPKR_HP	Dedicated power supply pins for the three amplifiers.	—
SRAM standby supply	VSTBY_SRAM	Standby voltage for the 128-kB internal SRAM	—
RTC standby supply	VSTBY_RTC	Standby voltage for the real-time clock module	—

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.

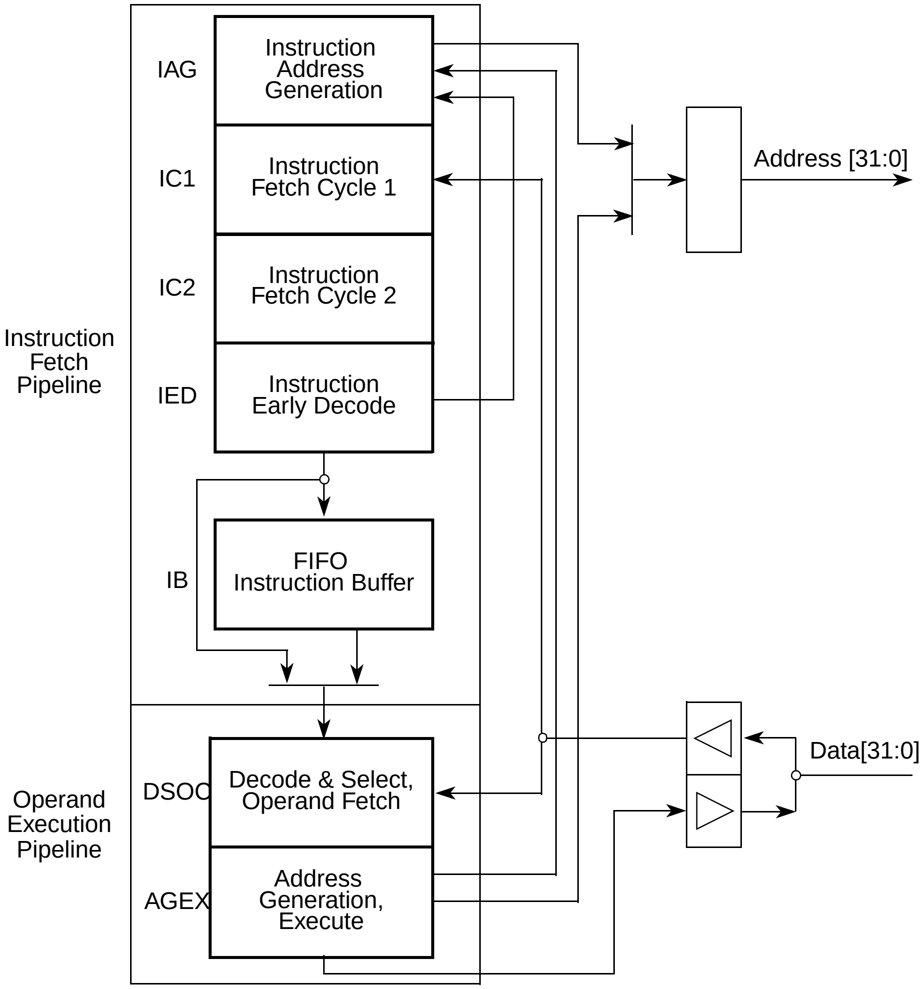

Instruction Fetch Pipeline
IAG
Instruction Address Generation
IC1
Instruction Fetch Cycle 1
IC2
Instruction Fetch Cycle 2
IED
Instruction Early Decode
IB
FIFO Instruction Buffer
Address [31:0]
Operand Execution Pipeline
DSOC
Decode & Select, Operand Fetch
AGEX
Address Generation, Execute
Data[31:0]

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), that decodes the instruction, fetches the required operands, and then executes the required function. Because the IFP and OEP pipelines are decoupled by an instruction buffer serving as a FIFO queue, the IFP is able to prefetch instructions in advance of their actual use by the OEP thereby minimizing time stalled waiting for instructions.

The 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, that consists of registers available in user mode as well as the following control registers:

- 16-bit status register (SR)
- 32-bit supervisor stack pointer (SSP)
- 32-bit vector base register (VBR)
- 32-bit cache control register (CACR)
- 32-bit access control registers (ACR0, ACR1)
- 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	0xCF3_64	No	3.2.1/3-6
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x0000_0690	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*)
Store: 0x180 + *n*; *n* = 0-7 (D*n*)

Access: User read/write
BDM read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Data																															
W																																
Reset (D2-D7)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Reset (D0, D1)	See Section 3.3.4.15, “Reset Exception”																															

Figure 3-2. Data Registers (D0–D7)

3.2.2 Address Registers (A0–A6)

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

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

Access: User read/write
BDM read/write

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

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

Access: A7: User or BDM read/write
OTHER_A7: Supervisor or BDM read/write

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

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

3.2.4 Condition Code Register (CCR)

The CCR is the LSB of the processor status register (SR). Bits 4–0 act as indicator flags for results generated by processor operations. The extend bit (X) is also an input operand during multiprecision arithmetic computations. The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.

BDM: LSB of Status Register (SR)

Access: User read/write
BDM read/write

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

Figure 3-5. Condition Code Register (CCR)

Table 3-2. CCR Field Descriptions

Field	Description
7 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 PC contents or places a new value in the PC. The PC is a base address for PC-relative operand addressing.

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

BDM: 0x80F (PC)

Access: User read/write
BDM read/write

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

Figure 3-6. Program Counter Register (PC)

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

BDM: 0x801 (VBR)

Access: Supervisor read/write
BDM read/write

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

Figure 3-7. Vector Base Register (VBR)

3.2.9 Status Register (SR)

The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits (CCR) are accessible. The control bits indicate the following states for the processor: trace mode (T bit), supervisor or user mode (S bit), and master or interrupt state (M bit). All defined bits in the SR have read/write access when in supervisor mode. The lower byte of the SR (the CCR) must be loaded explicitly after reset and before any compare (CMP), Bcc, or Scc instructions execute.

BDM: 0x80E (SR)

Access: Supervisor read/write
BDM read/write

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

Figure 3-8. Status Register (SR)

Table 3-3. SR Field Descriptions

Field	Description
15 T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	Reserved, must be cleared.

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:

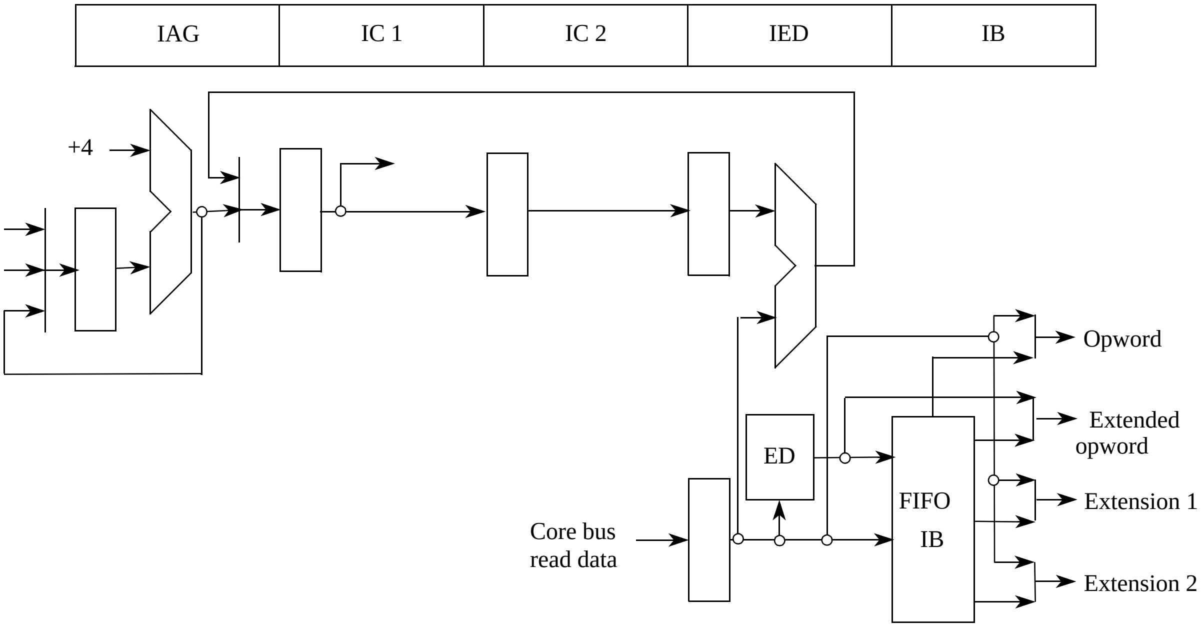


Figure 3-9. Version 3 ColdFire Processor Instruction Fetch Pipeline Diagram

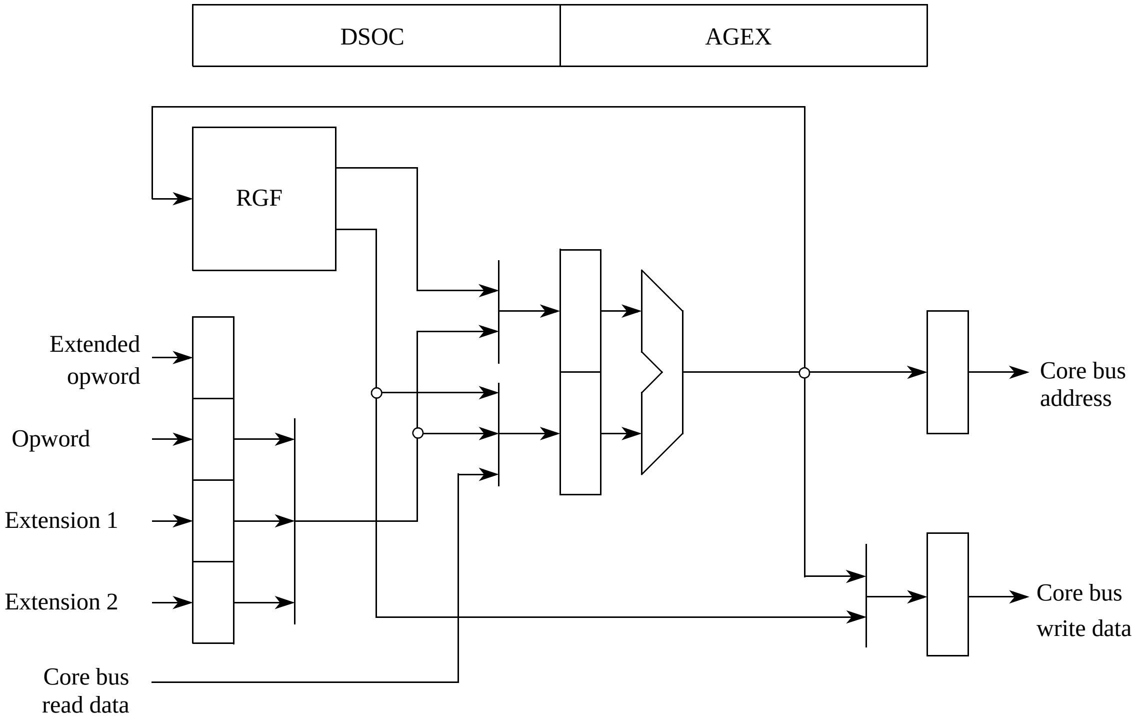


Figure 3-10. Version 3 ColdFire Processor Operand Execution Pipeline Diagram

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
STLDSR	Pushes the contents of the status register onto the stack and then reloads the status register with the immediate data value.

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 MB boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register. The index into the exception table is calculated as (4 × vector number). After the exception vector has been fetched, the vector contents determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has initiated, exception processing terminates and normal instruction processing continues in the handler.

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

The table contains 256 exception vectors; the first 64 are defined for the core and the remaining 192 are device-specific peripheral interrupt vectors. See Chapter 15, “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

Table 3-5. Exception Vector Assignments (continued)

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5	0x014	Fault	Divide by zero
6–7	0x018–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented line-F opcode
12	0x030	Next	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. In addition, the ISA_A+ architecture includes an instruction (STLDSR) that stores the current interrupt mask level and loads a value into the SR. This instruction is specifically intended for use as the first instruction of an interrupt service routine that services multiple interrupt requests with different interrupt levels. 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
00*xx*	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 is aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. 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 15, “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			
W																
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	CAU	0	0	ISA				DEBUG			
W																
Reset	0	1	1	0	0	1	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.
11	Reserved.
10 CAU	Cryptographic acceleration unit present. This bit signals if the optional cryptographic acceleration unit (CAU) is present in the processor core. 0 CAU coprocessor engine not present in core. 1 CAU coprocessor engine is present in core. (This is the value used for this device.)
9–8	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)
Store: 0x181 (D1)

Access: User read-only
BDM read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CLSZ		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	1	0	0	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 KB 0011 2 KB 0100 4 KB 0101 8 KB 0110 16 KB (This is the value used for this device) 0111 32 KB Else Reserved for future use

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

Field	Description
7–3 SRAMSZ	SRAM bank size. 00000 No SRAM 00010 512 bytes 00100 1 KB 00110 2 KB 01000 4 KB 01010 8 KB 01100 16 KB 01111 24 KB 01110 32 KB 10000 64 KB 10010 128 KB (This is the value used for this device) 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)	—	—	—	—	—	—	—
STLDSR	#imm	—	—	—	—	—	—	—	5(0/1)
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)	—	—	—	—	—	—	—

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, MACSR	5(0/0)	—	—	—	—	—	—	5(0/0)
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).

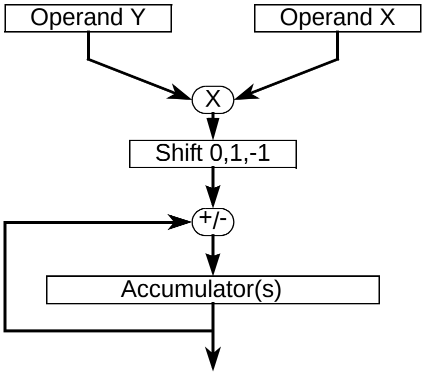


Figure 4-1. Multiply-Accumulate Functionality Diagram

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAV*n*				OMC	S/U	F/I	R/T	N	Z	V	EV
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 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 forms the general overflow flag, MACSR[V]. Once set, each flag remains set until V is cleared by a `move.l, MACSR` instruction or the accumulator is loaded directly. Bit 11: Accumulator 3 ... Bit 8: Accumulator 0

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>y&,Rw
```

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

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

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

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

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

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

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

BDM: 0x805 (MASK)

Access: User read/write
BDM read/write

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

Figure 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)
0x809 (ACC1)
0x80A (ACC2)
0x80B (ACC3)

Access: User read/write
BDM read/write

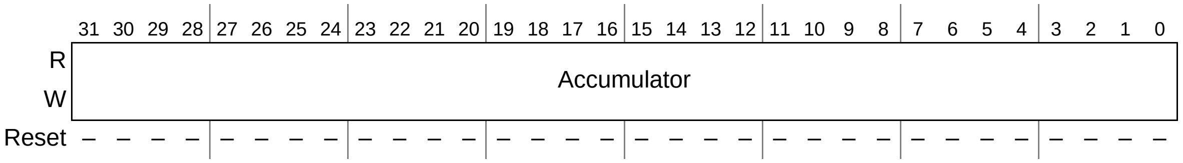


Figure 4-4. Accumulator Registers (ACC0–3)

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

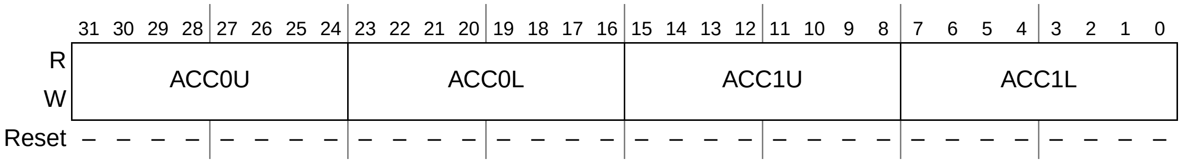


Figure 4-5. Accumulator Extension Register (ACCext01)

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 W	ACC2U								ACC2L								ACC3U								ACC3L							
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.

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

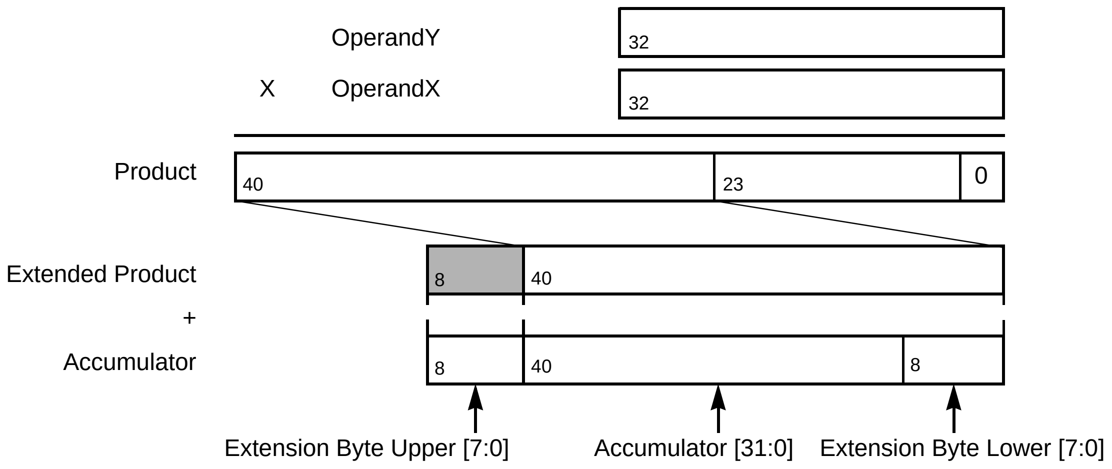


Figure 4-7. Fractional Alignment

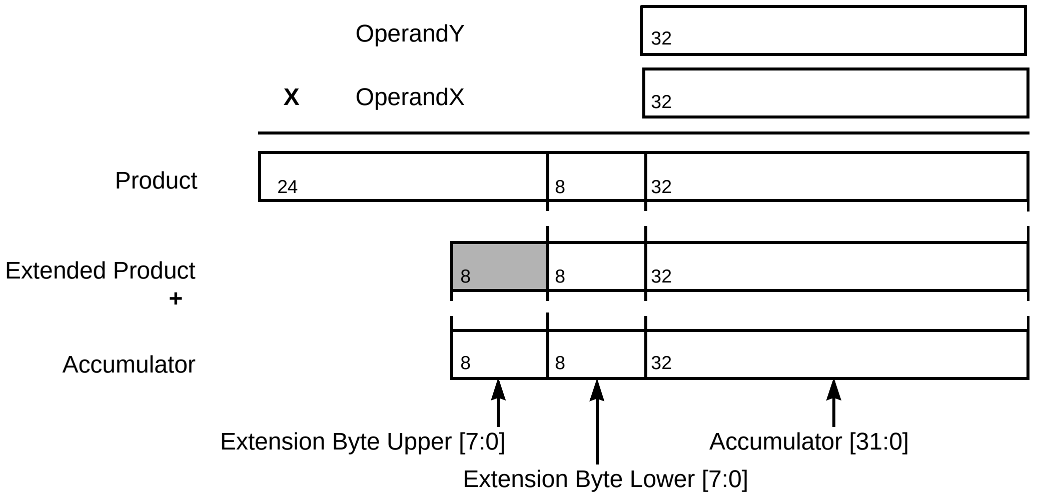


Figure 4-8. Signed and Unsigned Integer Alignment

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 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.

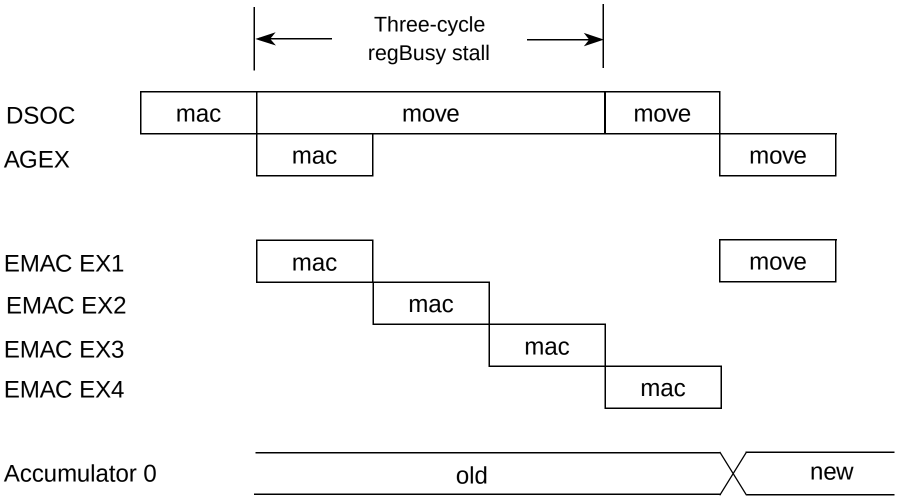


Figure 4-9. EMAC-Specific OEP Sequence Stall

In Figure 4-9, the OEP stalls the store-accumulator instruction for three cycles: the EMAC pipleline 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 $-2^{(N-1)} \leq \text{operand} \leq 2^{(N-1)} - 1$. The binary point is right of the lsb.
2. Unsigned integer: In this format, an N-bit operand value lies in the range $0 \leq \text{operand} \leq 2^N - 1$. The binary point is right of the lsb.
3. Two's complement, signed fractional: In an N-bit number, the first bit is the sign bit. The remaining bits signify the first N-1 bits after the binary point. Given an N-bit number, $a_{N-1}a_{N-2}a_{N-3}\ldots a_2a_1a_0$, its value is given by the equation in Equation 4-3.

$$\text{value} = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{-(i+1-N)} \cdot ai \qquad \textit{Eqn. 4-3}$$

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

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

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

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

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

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

            /* check for accumulation overflow */
            if (accumulationOverflow == 1)
               then {MACSR.PAVn = 1
                     MACSR.V = 1
                     if (MACSR.OMC == 1)
                        then /* accumulation overflow,
                                saturationMode enabled */
                             if (result[47] == 1)
                                then result[47:0] = 0x0000_7fff_ffff
                                else result[47:0] = 0xffff_8000_0000
               }
            /* transfer the result to the accumulator */
            ACCx[47:0] = result[47:0]
      }
      MACSR.V = MACSR.PAVn
      MACSR.N = ACCx[47]
      if (ACCx[47:0] == 0x0000_0000_0000)
         then MACSR.Z = 1
         else MACSR.Z = 0
      if ((ACCx[47:31] == 0x0000_0) || (ACCx[47:31] == 0xffff_1))
         then MACSR.EV = 0
         else MACSR.EV = 1
break;
   case 1,3:              /* signed fractionals */
   if (MACSR.OMC == 0 || MACSR.PAVn == 0)
      then {
            MACSR.PAVn = 0
            if (sz == word)
               then {if (U/Ly == 1)
                        then operandY[31:0] = {Ry[31:16], 0x0000}
                        else operandY[31:0] = {Ry[15:0],  0x0000}
                     if (U/Lx == 1)
```

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

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

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

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

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

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

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

```
                                     result[47:0] = 0xffff_ffff_ffff
                  }

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

Chapter 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.

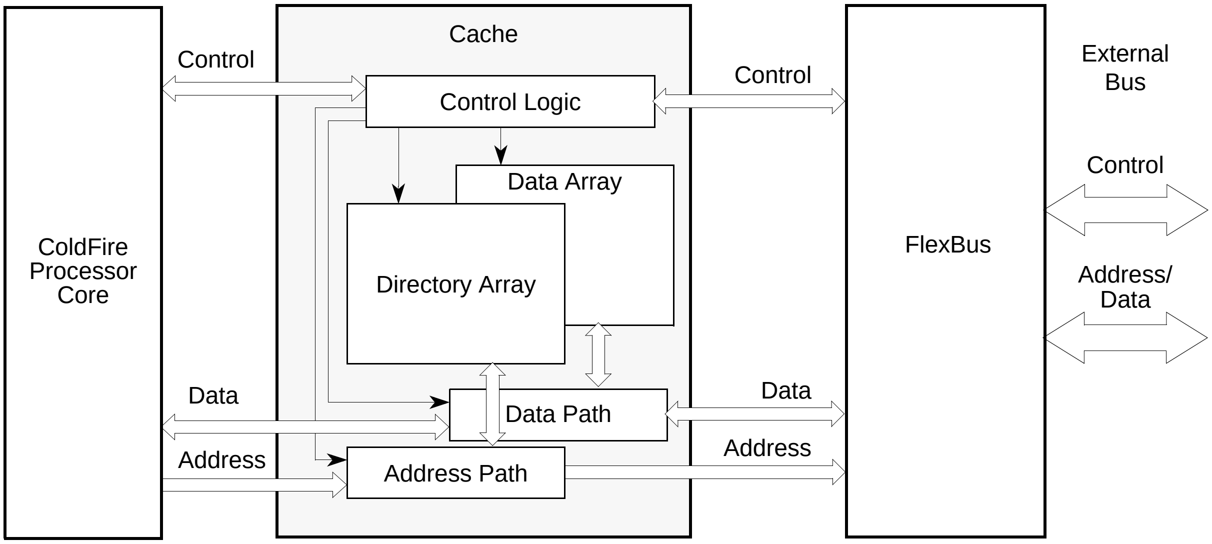


Figure 5-1. Unified Cache Organization

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

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EC	0	ESB	DPI	HLCK	0	0	CINVA	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	DNFB	DCM		0	0	DW	EUSP	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

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. Before using the cache line fill buffer, clear the cache by setting CINVA. **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.

Table 5-2. CACR Field Descriptions (continued)

Field	Description
5 DW	Default write protect. Indicates the default write privilege. 0 Read and write accesses permitted 1 Write accesses not permitted
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 (ACR*n*) 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 ACR*n* registers must define the non-cacheable attribute for this address space.

BDM: 0x004 (ACR0)
0x005 (ACR1)

Access: MOVEC write-only
Debug read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Address Base								Address Mask								E	S		0	0	0	0	0	0	CM		0	0	W	0	0
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	0	–	–	0	0	0	0	0	0	–	–	0	0	–	0	0

Figure 5-3. Access Control Register Format (ACR*n*)

Table 5-3. ACR*n* 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 ACR*n* bits. 0 Access control attributes disabled 1 Access control attributes enabled

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

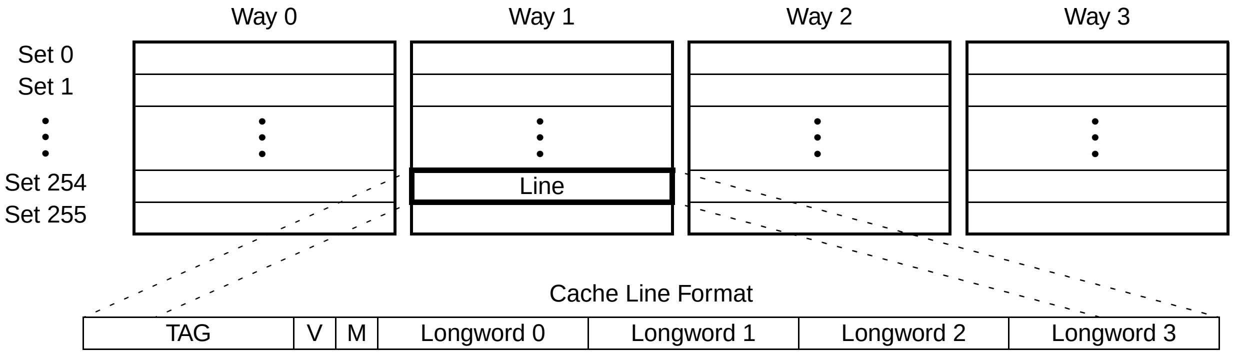
Field	Description
14–13 S	Supervisor mode. Specifies whether only user or supervisor accesses are allowed in this address range or if the type of access is a don’t care: 00 Match addresses only in user mode 01 Match addresses only in supervisor mode 1X Execute cache matching on all accesses
12–7	Reserved, 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.



Where:
TAG—20-bit address tag
V—Valid bit for line
M—Modified bit for line

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.”

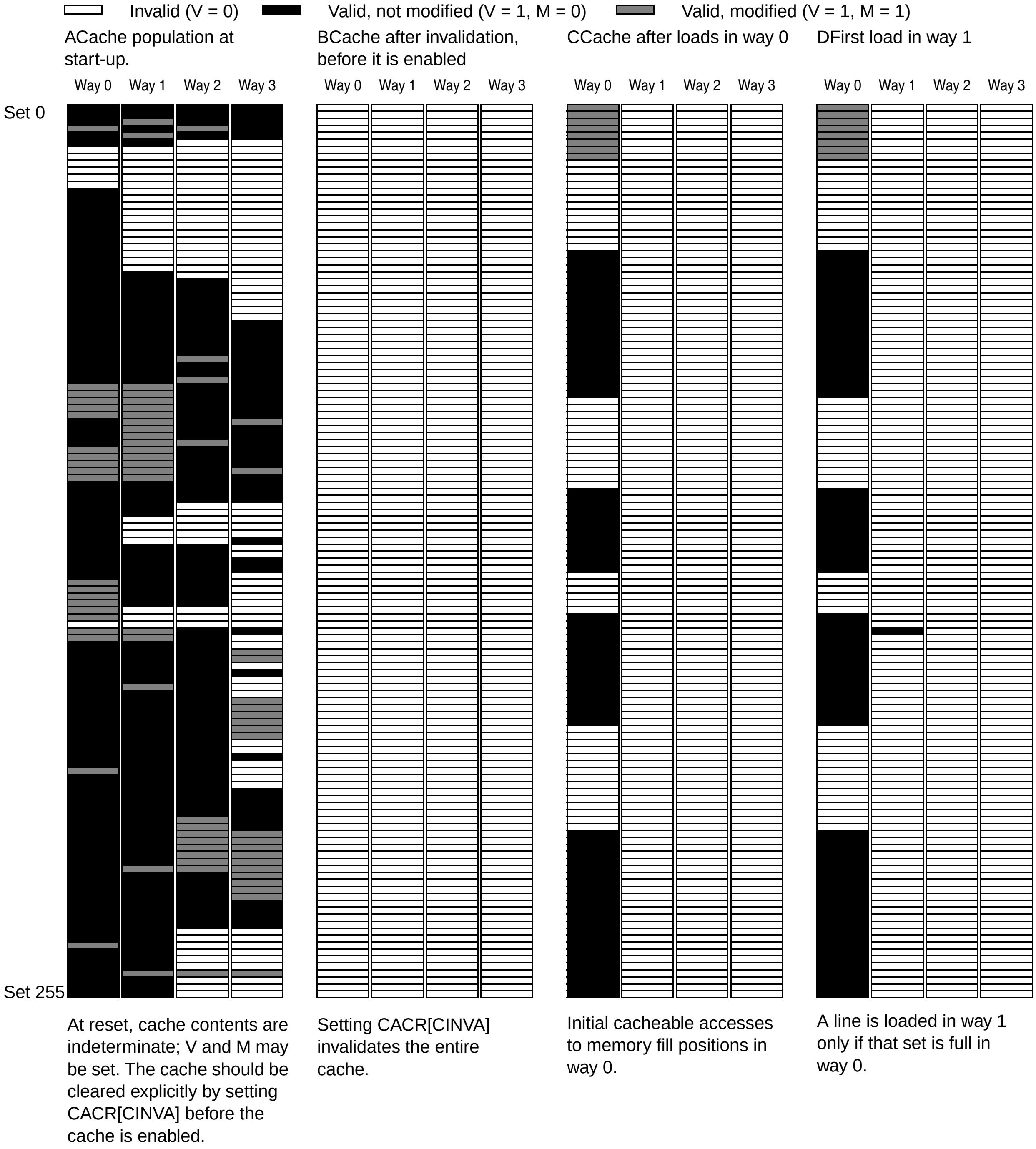

Invalid (V = 0)
Valid, not modified (V = 1, M = 0)
Valid, modified (V = 1, M = 1)
ACache population at start-up.
BCache after invalidation, before it is enabled
CCache after loads in way 0
DFirst load in way 1
Way 0
Way 1
Way 2
Way 3
Set 0
Set 255
At reset, cache contents are indeterminate; V and M may be set. The cache should be cleared explicitly by setting CACR[CINVA] before the cache is enabled.
Setting CACR[CINVA] invalidates the entire cache.
Initial cacheable accesses to memory fill positions in way 0.
A line is loaded in way 1 only if that set is full in way 0.

Figure 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.

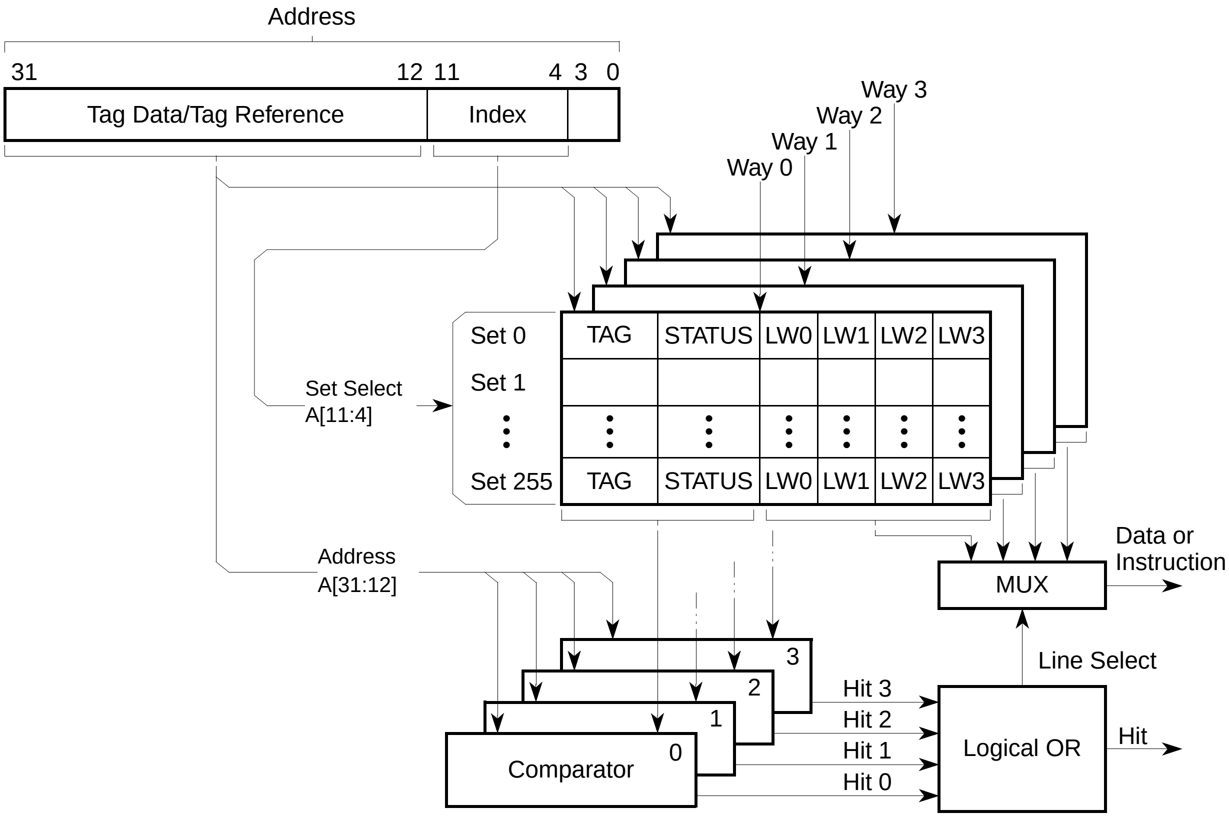


Figure 5-6. Data Caching Operation

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 ACR*n*[CM] bit corresponding to the address of the access specifies the caching modes. If an address does not match an ACR, the default caching mode is defined by CACR[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 ACR*n*[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 ACR*n*[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 ACR*n*[CM] indicates cache-inhibited mode, precise or imprecise, the controller bypasses the cache and performs an external transfer. If a line in the cache matches the address and the mode is cache-inhibited, the cache does not automatically push the line if it is modified, nor does it invalidate the line if it is valid. Before switching cache mode, execute a CPUSHL instruction or set CACR[CINVA] to invalidate the entire cache.

If ACR*n*[CM] indicates precise mode, the sequence of read and write accesses to the region is guaranteed to match the instruction sequence. In imprecise mode, the processor core allows read accesses that hit 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 ACR*n*[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

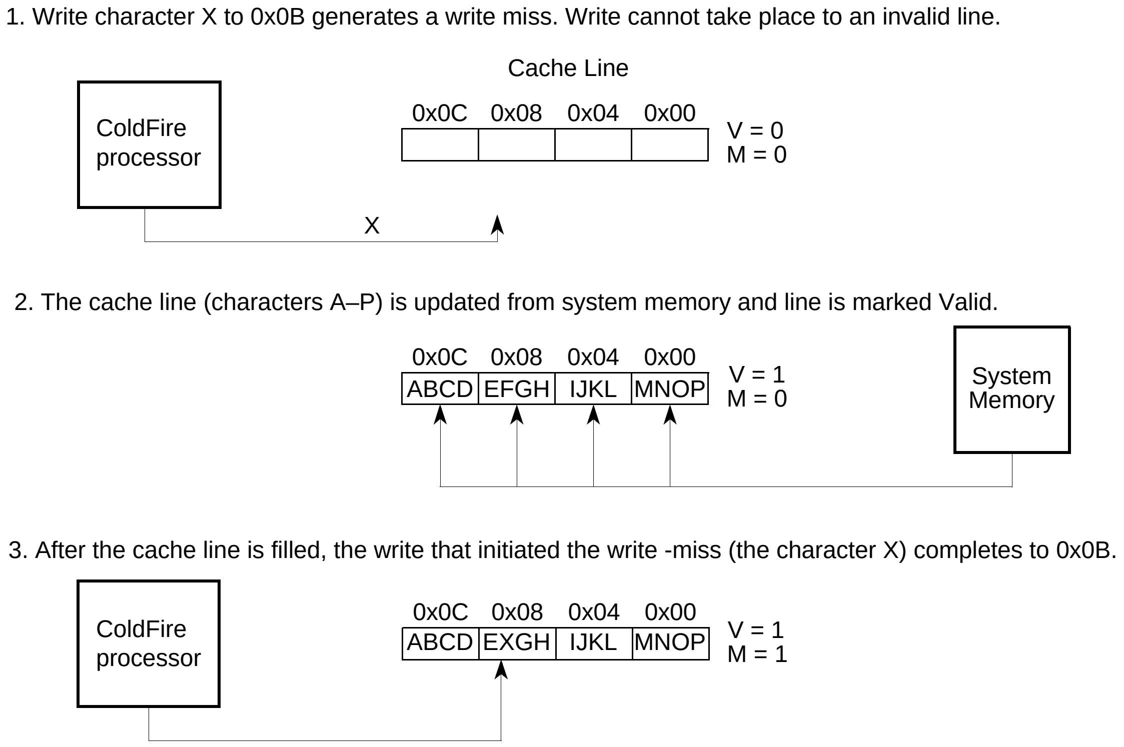
The following sections describe the cache protocol for processor accesses and assumes that the data is cacheable (that is, write-through or copyback).

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 18, “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.

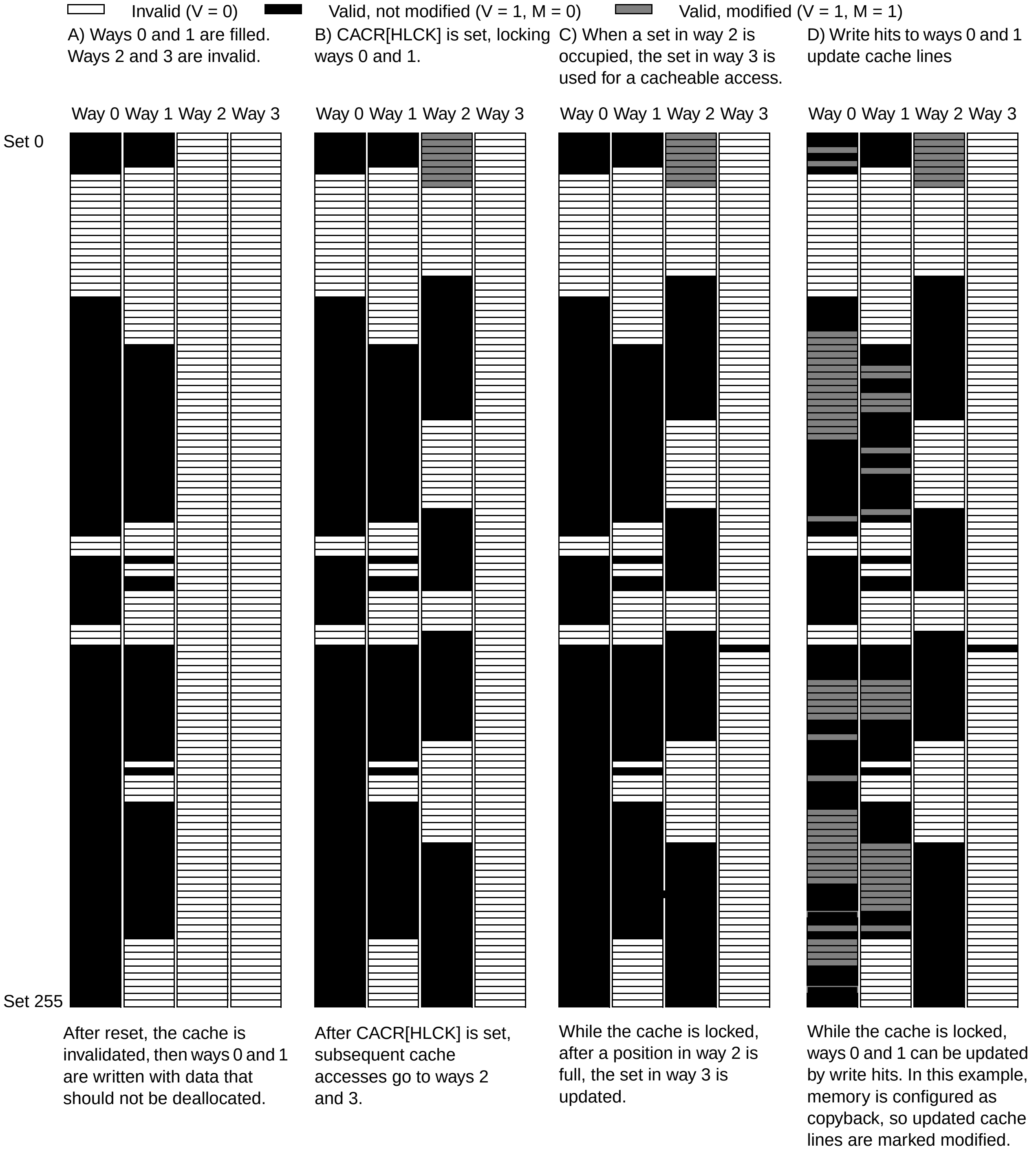

Invalid (V = 0)
Valid, not modified (V = 1, M = 0)
Valid, modified (V = 1, M = 1)
A) Ways 0 and 1 are filled. Ways 2 and 3 are invalid.
B) CACR[HLCK] is set, locking ways 0 and 1.
C) When a set in way 2 is occupied, the set in way 3 is used for a cacheable access.
D) Write hits to ways 0 and 1 update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

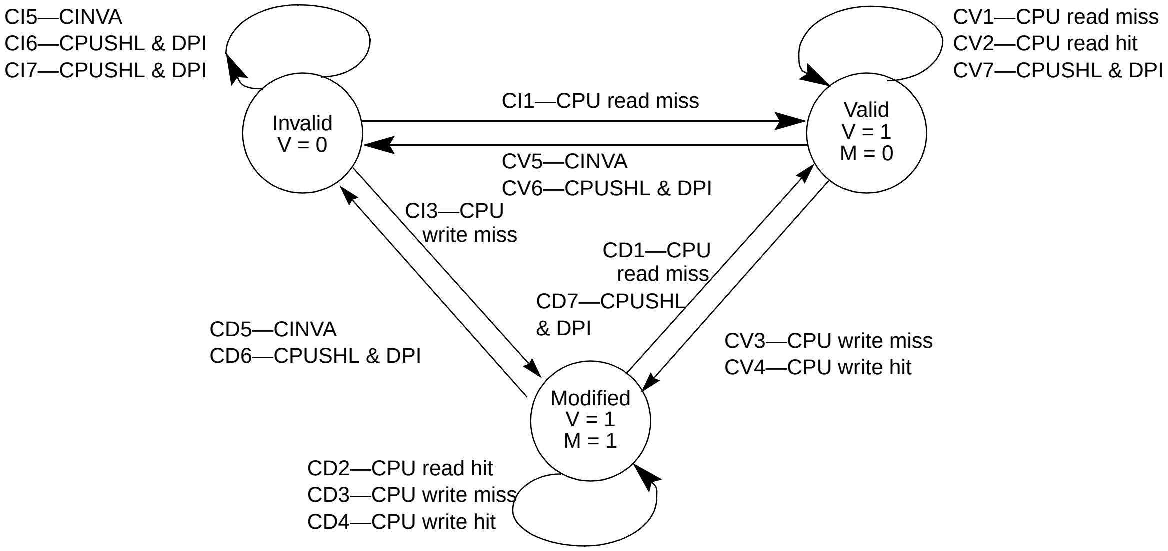


Figure 5-10. Cache Line State Diagram—Copyback Mode

Figure 5-11 shows the two possible states for a cache line in write-through mode.

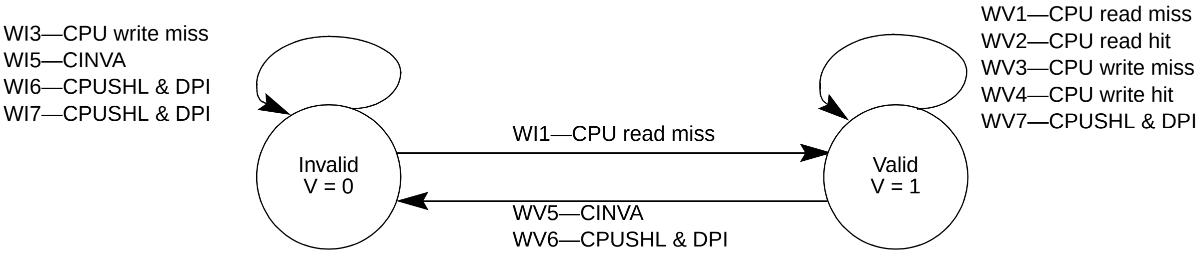


Figure 5-11. Cache Line State Diagram—Write-Through Mode

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-128K 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 provideaccess 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 128 Kbyte SRAM
- Single-cycle access
- Physically located on the processor's high-speed local bus
- Memory location programmable on any 0-modulo-128 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 – 0x8FFE_0000. The adress must be 0-modulo-128 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 0x8002_0000 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	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	0	0	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
31–17 BA	Base Address. Defines the 0-modulo-128K base address of the SRAM module. By programming this field, the SRAM may be located on any 128-Kbyte boundary within the processor's 256-Mbyte address space. For proper operation, the base address must be set to between 0x8000_0000 and 0x8FFE_0000.
16–12	Reserved, must be cleared.
11–10 PRIU PRIL	Priority Bit. PRIU determines if the SRAM backdoor or CPU has priority in the upper 128K bank of memory. PRIL determines if the SRAM backdoor or CPU has priority in the lower 128K bank of memory. If a bit is set, the CPU has priority. If a bit is cleared, the SRAM backdoor has priority. Priority is determined according to the following table: **PRIU,PRIL** / **Upper Bank Priority** / **Lower Bank Priority** 00 / SRAM Backdoor / SRAM Backdoor 01 / SRAM Backdoor / CPU 10 / CPU / SRAM Backdoor 11 / CPU / CPU **Note:** The recommended setting (maximum performance) for the priority bits is 00.
9 BDE	Backdoor Enable. Allows access by non-core bus masters via the SRAM backdoor on the crossbar switch 0 Non-core crossbar switch master access to memory is disabled. 1 Non-core crossbar switch master access to memory is enabled.
8 WP	Write Protect. Allows only read accesses to the SRAM. When this bit is set, any attempted write access from the core generates an access error exception to the ColdFire processor core. 0 Allows core 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.

Table 6-2. RAMBAR Field Descriptions (continued)

Field	Description
5–1 C/I, SC, SD, UC, UD	Address Space Masks (AS*n*). These five bit fields allow types of accesses to be masked or inhibited from accessing the SRAM module. The address space mask bits are: C/I = CPU space/interrupt acknowledge cycle mask SC = Supervisor code address space mask SD = Supervisor data address space mask UC = User code address space mask UD = User data address space mask For each address space bit: 0 An access to the SRAM module can occur for this address space 1 Disable this address space from the SRAM module. If a reference using this address space is made, it is inhibited from accessing the SRAM module and is processed like any other non-SRAM reference. These bits do not affect accesses by non-core bus masters using the SRAM backdoor port in any manner. These bits are useful for power management as detailed in Section 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   #32768,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 AS*n* bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. Table 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

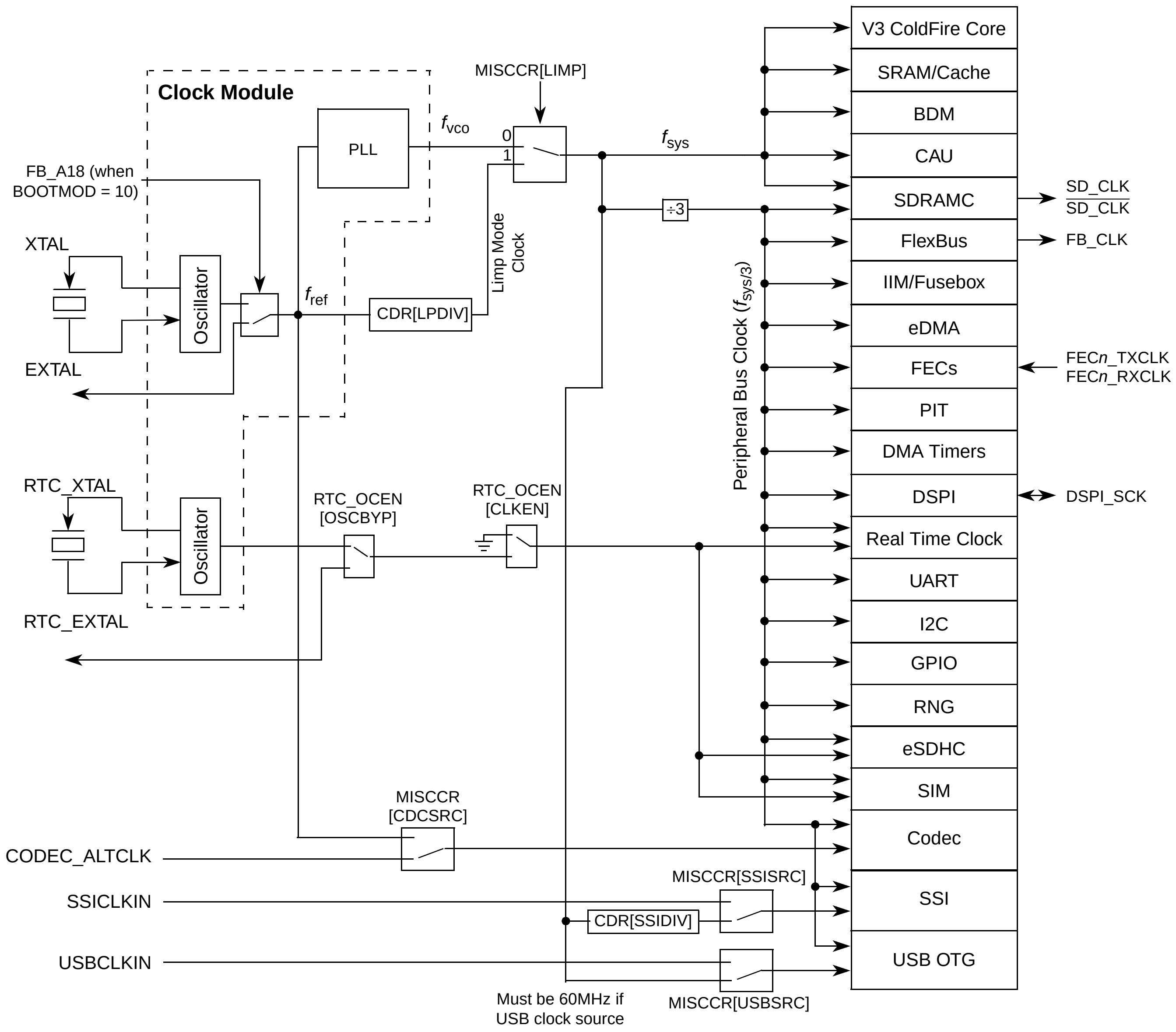
The clock module allows the device to be configured for one of several clocking methods. Clocking modes include internal phase-locked loop (PLL) clocking with an external clock reference or an external crystal reference supported by an internal crystal amplifier. The PLL can also be disabled, and an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

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

NOTE

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

Figure 7-1 is a high-level representation of clock connections. The exact functionality of the blocks is not illustrated (clocks to the SDRAMC and FECs are disabled when the device is in limp mode, and the clocks to individual modules may be disabled via the peripheral power management registers as described in Chapter 8, “Power Management”).



Notes:

[1] The SDRAMC and FECs are disabled in limp mode. The USB controllers are essentially disabled, as they are able to capture a wake-up event to bring the device out of limp mode.

[2] The SDRAMC, SSI, USB, codec, and real time clock contain some logic that uses the $f_{sys/3}$ clock, in addition to the module-specific clock.

Figure 7-1. Device Clock Connections

7.1.1 Block Diagram

Figure 7-2 shows the clock module block diagram.

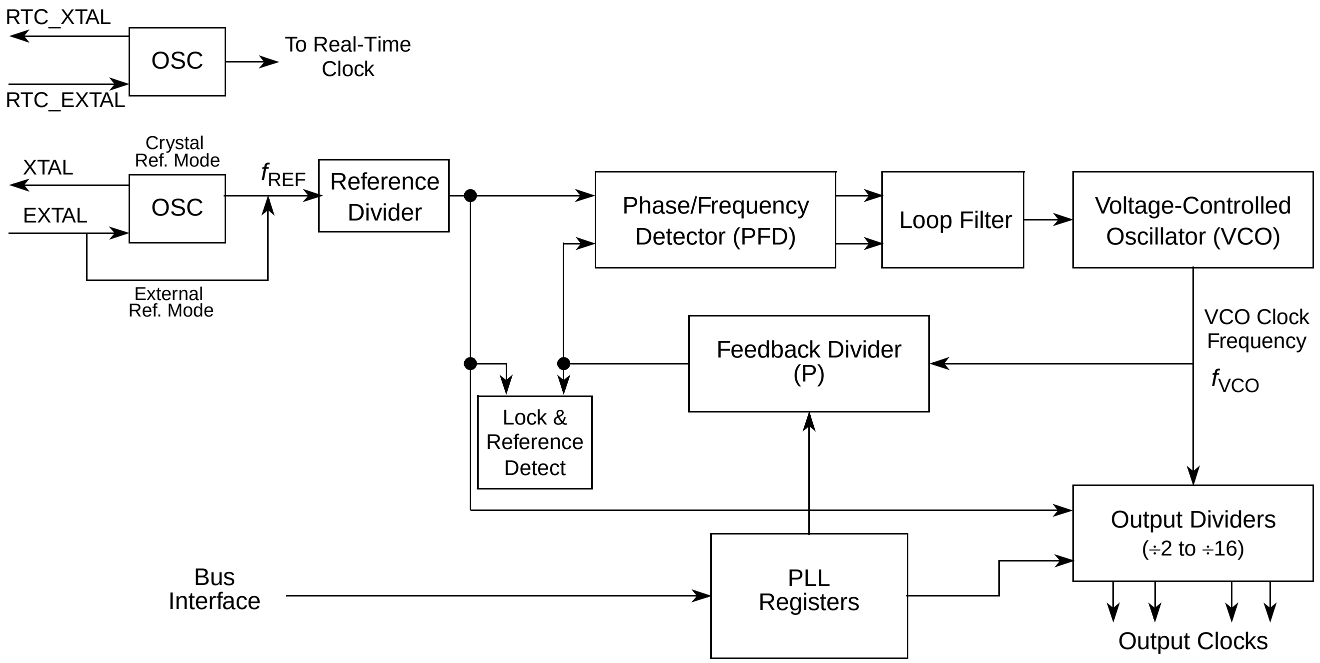


Figure 7-2. Clock Module Diagram

7.1.2 Features

Features of the clock module include:

- 14–40 MHz input clock
- Programmable frequency multiplication factor settings generating voltage-controlled oscillator (VCO) frequencies from 240–480 MHz, resulting in a core frequency of 15 MHz (f_{vco} ÷ 16) to 240 MHz (maximum rated frequency).
- Four user-programmable output dividers to produce the following clocks
  - — Core clock (f_{sys})
  - — Bus and inverted bus clocks ($f_{sys/3}$)
  - — SDRAM controller clock ($f_{sys/3} \times 2$)
  - — USB clock (60 MHz)
- Loss-of-lock detection and reset
- Loss-of-clock detection and reset
- Support for low-power modes
- Direct clocking of system by input clock, bypassing the PLL
- 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 $\overline{\text{RSTOUT}}$ asserts until it negates. Refer to Chapter 9, "Chip Configuration Module (CCM)", for more details.

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 (14–40 MHz) from the crystal oscillator circuit and multiplies the frequency to create the PLL output clock. It can synthesize frequencies ranging from 22–33.75x the input frequency.

You must supply a crystal oscillator within the appropriate input frequency range, the crystal manufacturer's recommended external support circuitry, and short signal route from the device to the crystal.

7.1.3.2 Normal PLL Mode with External Reference

This second mode is the 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 (oscillator bypass mode).

7.1.3.3 Input Clock (Limp) Mode

After reset or by setting MISCCR[LIMP] bit, the device is placed into a low-frequency limp mode, in which the PLL is placed in reset 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]. For more information on programming the divider, see Chapter 9, "Chip Configuration Module (CCM)". 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 reset mode to reduce overall system power consumption. A 3:1 ratio is still 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 FECs are not functional in limp mode. Also, the USB controllers must source their system clocks through alternate, external sources.

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

Entering limp mode via the MISCCR[LIMP] bit requires a special procedure for the SDRAM module. As noted above, the SDRAM controller is disabled in limp mode, so follow these two critical steps before setting the MISCCR[LIMP] bit:

1. Code execution must be transferred to another memory resource. Primary options are:
   - Memory device attached to the FlexBus boot chip-select
   - On-chip SRAM (but not the CPU cache, as it may have to be flushed upon limp mode entrance or exit)
2. The SDRAM controller must be placed in self-refresh mode to avoid data loss while the SDRAMC shuts down.

7.1.3.4 Low-power Mode Operation

This subsection describes the clock module operation in low-power and halted modes of operation. Low-power modes are described in Chapter 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. CPU is stopped.	Clock module does not cause exit, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only. CPU is stopped.	Clock module does not cause exit, but normal clocking resumes upon mode exit
Stop	All system clocks disabled	Clock module does not cause exit, but clock sources are re-enabled and normal clocking resumes upon mode exit
Halted	Normal	Clock module does not cause exit

In wait and doze modes, the system clocks to the peripherals are enabled, and the clocks to the core and SRAM are stopped. Each module can disable its clock locally at the module level.

In stop mode, all system clocks are disabled (except the real-time clock which continues to run via its external clock). There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wake-up recovery time. The PLL can be disabled in stop mode, but requires a wake-up period before it relocks. The oscillator can also be disabled during stop mode, but it requires a wake-up period to restart.

When the PLL is enabled in stop mode (LPCR[STPMD] = 00), the external FB_CLK signal can support systems using FB_CLK as the clock source. For more information about operating the PLL in stop mode, see Section 8.2.5, “Low-Power Control Register (LPCR)”.

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

7.2 Memory Map/Register Definition

The PLL programming model consists of the following:

Table 7-2. PLL Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_0000	PLL Control Register (PLL_CR)	32	R/W	0x0000_101D	7.2.1/7-6
0xFC0C_0004	PLL Divider register (PLL_DR)	32	R/W	0x0000_7251	7.2.2/7-7
0xFC0C_0008	PLL Status Register (PLL_SR)	32	R/W	0x0000_0000	7.2.3/7-9

7.2.1 PLL Control Register (PLL_CR)

The PLL_CR register controls the feedback and reference dividers for generating the core and bus clocks. For details on altering these values after reset, see Section 7.3.1, “PLL Frequency Multiplication Factor Select.” PLL_CR also contains control bits for loss-of-clock and loss-of-lock detection.

Address: 0xFC0C_0000 (PLL_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	LOC IRQ	LOC RE	LOC EN
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	LOL IRQ	LOL RE	LOL EN	0	REFDIV			0	0	FBKDIV					
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	1

Figure 7-3. PLL Control Register (PLL_CR)

Table 7-3. PLL_CR Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18 LOCIRQ	Loss of clock interrupt request. Determines how the processor handles a loss-of-clock condition when LOCEN is set. If LOCEN is cleared, this bit has no effect. If PLL_SR[LOCF] is set, setting LOCIRQ causes an immediate interrupt request. 0 Ignore loss of clock, interrupt not requested 1 Request interrupt on loss of clock
17 LOCRE	Loss of clock reset enable. If LOCEN is set, determines how the processor handles a loss-of-clock condition. If LOCEN is cleared, this bit has no effect. Setting LOCRE causes an immediate reset request upon detection of loss of clock. 0 Ignore loss of clock, reset not requested 1 Request reset on loss of clock
16 LOCEN	Loss of clock enable. Enables the loss-of-clock feature. 0 Loss of clock disabled 1 Loss of clock enabled

Table 7-3. PLL_CR Field Descriptions (continued)

Field	Description
15	Reserved, must be cleared.
14 LOLIRQ	Loss of lock interrupt request. If LOLEN is set, determines how the processor handles a loss-of-lock condition. If LOLEN is cleared, this bit has no effect. The PLL must be locked when LOLIR is enabled to avoid an immediate interrupt request. 0 Ignore loss of lock, interrupt not requested 1 Request interrupt on loss of lock
13 LOLRE	Loss of lock reset enable. If LOLEN is set, determines how the processor handles a loss-of-lock condition. If LOLEN is cleared, this bit has no effect. The PLL must be locked when LOLRE is enabled to avoid an immediate reset request. 0 Ignore loss of lock, reset not requested 1 Request reset on loss of lock
12 LOLEN	Loss of lock enable. Enables the loss-of-lock feature. 0 Loss of lock disabled 1 Loss of lock enabled **Note:** Only change this bit when the device is in limp mode.
11	Reserved, must be cleared.
10–8 REFDIV	Reference divider setting. Along with FBKDIV, determines the VCO clock frequency. See the FBKDIV field for details. 000 1 001 2 010 Reserved 011 Reserved 100 Reserved Else Reserved. A divider value of one is used.
7–6	Reserved, must be cleared.
5–0 FBKDIV	Feedback divider setting. Along with REFDIV, determines the VCO clock frequency. The feedback divider is the value of this bit field plus 1. The resulting VCO frequency is shown below: $f_{VCO} = \frac{f_{REF} \times (\mathrm{FBKDIV} + 1)}{2^{\mathrm{REFDIV}}}$ ***Eqn. 7-1***

7.2.2 PLL Divider Register (PLL_DR)

The PLL_DR register controls the output dividers for generating the core, bus, and USB clocks. For details on altering these values after reset, see Section 7.3.1, “PLL Frequency Multiplication Factor Select”.

NOTE

A single longword (32-bit) write to the PLL_DR register is required. If back-to-back word or longword writes are attempted, some of the clocks in the system change frequency before others, which can cause the device to hang.

Address: 0xFC0C_0004 (PLL_DR)   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	OUTDIV4				OUTDIV3				OUTDIV2				OUTDIV1			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	1	0	0	0	1

Figure 7-4. PLL Divider Register (PLL_DR)

Table 7-4. PLL_DR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–16</td><td>Reserved, must be cleared.</td></tr>
<tr><td>15–12
OUTDIV4</td><td>Output divider for generating the USB controllers clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock.

$$f_{USB} = \frac{f_{VCO}}{OUTDIV4 + 1} \qquad \textit{Eqn. 7-2}$$

Note: If used as the USB clock source, the OUTDIV4 resulting frequency must be 60 MHz.</td></tr>
<tr><td>11–8
OUTDIV3</td><td>Output divider for generating the SDRAM controller clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock.

$$f_{SDRAM} = \frac{f_{VCO}}{OUTDIV3 + 1} \qquad \textit{Eqn. 7-3}$$

Note: The OUTDIV3 resulting frequency must be two times the OUTDIV2 resulting frequency.</td></tr>
<tr><td>7–4
OUTDIV2</td><td>Output divider for generating the internal bus clock frequency, including the FlexBus clock (FB_CLK). The divider is the value of this bit field plus 1. A value of zero disables this clock.

$$f_{SYS/3} = \frac{f_{SYS}}{3} = \frac{f_{VCO}}{OUTDIV2 + 1} \qquad \textit{Eqn. 7-4}$$

Note: If the core clock is enabled (OUTDIV1 ≠ 0), the internal bus clock frequency must be one-third the core clock frequency. Therefore, the valid OUTDIV2 settings are limited to 0, 5, 8, 11, and 14. See OUTDIV1 bit description for more details.</td></tr>
<tr><td>3–0
OUTDIV1</td><td>Output divider for generating the core clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock.

$$f_{SYS} = \frac{f_{VCO}}{OUTDIV1 + 1} \qquad \textit{Eqn. 7-5}$$

Note: If the internal bus clock is enabled, the core clock frequency must be three times the internal bus clock frequency. Therefore, the valid OUTDIV1 values are limited to 0x0–0x4.

Note: If the internal bus clock, SDRAM clock, and core clock are enabled, there are only two valid settings for OUTDIV1–3 as shown in the table below:

<table>
<tr><th></th><th colspan="2">OUTDIV1</th><th colspan="2">OUTDIV2</th><th colspan="2">OUTDIV3</th></tr>
<tr><td><b>Valid setting #1</b></td><td>0001</td><td>÷ 2</td><td>0101</td><td>÷ 6</td><td>0010</td><td>÷ 3</td></tr>
<tr><td><b>Valid setting #2</b></td><td>0011</td><td>÷ 4</td><td>1011</td><td>÷ 12</td><td>0101</td><td>÷ 6</td></tr>
</table>
</td></tr>
</table>

7.2.3 PLL Status Register (PLL_SR)

The PLL status register provides indicators for PLL operating mode, loss-of-lock, and loss-of-clock status. You must write a one to the loss-of-lock and loss-of-clock flag bits to clear them. All other bits in the PLL_SR are read-only.

Address: 0xFC0C_0008 (PLL_SR) 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	LOCF	LOC	0	LOLF	LOCKS	LOCK	0	MODE		
W							w1c			w1c						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-5. PLL Status Register (PLL_SR)

Table 7-5. PLL_SR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9 LOCF	Loss-of-clock flag. If PLL_CR[LOCIRQ, LOCEN] are set, LOCF is set upon loss-of-clock detection and is subsequently used to generate the interrupt request. To clear LOCF, assert reset or write a one to this bit. Writing zero has no effect. This bit is sticky; if clocks are restored, the bit remains set until writing a one or asserting reset. 0 Interrupt service not requested 1 Interrupt service requested
8 LOC	Loss-of-clock status. Indicates whether the PLL has lost its reference clock. If you read this bit at the same time the loss-of-clock condition occurs, the bit does not reflect the current loss-of-clock condition. If a loss-of-clock condition occurs which sets this bit and the clocks later return to normal, this bit is cleared. A loss-of-clock condition can only be detected if PLL_CR[LOCEN] is set. 0 Clocks are operating normally 1 Clocks have failed due to a reference failure
7	Reserved, must be cleared.
6 LOLF	Loss-of-lock flag. If PLL_CR[LOLIR, LOLEN] are set, LOLF is set upon loss of lock and is subsequently used to generate the interrupt request. To clear LOLF, assert reset or write a one to this bit. Writing 0 has no effect. This bit is sticky; if lock is reacquired, the bit remains set until writing a one or asserting reset. 0 Interrupt service not requested 1 Interrupt service requested
5 LOCKS	Sticky PLL lock status bit. Set by the lock-detect circuitry when the PLL acquires lock. If the PLL loses lock, this bit is cleared and remains cleared even after the PLL re-locks. If you read this bit when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition. 0 PLL has lost lock since last reset 1 PLL has not lost lock since last reset

Table 7-5. PLL_SR Field Descriptions (continued)

Field	Description
4 LOCK	PLL lock status bit. Indicates if the PLL has acquired lock. If PLL_CR[LOLEN] is set, PLL lock occurs when the synthesized frequency matches to within 0.75% of the programmed frequency. The PLL loses lock when a frequency deviation of greater than ~1.5% occurs. If PLL_CR[LOLEN] is cleared, PLL lock is declared following 2000 reference clock cycles after negation of reset. The LOCK bit remains set until reset is asserted. In functional mode, LOCK is asserted following 16 reference clock cycles after negation of reset. The LOCK bit remains set until reset is asserted. If you read this bit when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition. 0 PLL is unlocked 1 PLL is locked
3	Reserved, must be cleared.
2–0 MODE	PLL mode setting. Read-only bit indicating the current PLL mode. 000 PLL functional mode Else Reserved

7.3 Functional Description

This subsection provides a functional description of the clock module.

7.3.1 PLL Frequency Multiplication Factor Select

The frequency multiplication factor of the PLL is defined by the feedback, reference, and output dividers specified in the PLL_CR and PLL_DR registers. An example equation for the core frequency is given below:

$$f_{\mathrm{SYS}} = \frac{f_{\mathrm{VCO}}}{\mathrm{PLL\_DR[OUTDIV1]} + 1} \qquad \textbf{\textit{Eqn. 7-6}}$$

where f_{sys} is the clock frequency of the ColdFire core and f_{REF} is the PLL clock source as shown in Figure 7-1. The allowable range of values for OUTDIV*n* is 1–15. The other clocks on the processor are configurable in a similar fashion. However, there are various dependencies. See Section 7.2.2, “PLL Divider Register (PLL_DR),” for details.

The PLL_DR[OUTDIV*n*] fields can be changed during normal operation or when the device is in limp mode. After a new value is written to the PLL_DR, the PLL synchronizes the new value of the PLL_DR with the VCO clock domain. Then, the transition from the old divider value to the new divider value takes place, such that the PLL output clocks remain glitch-free. During the adjustment to the new divider value, a PLL output clock may experience an intermediate transition while the divider values are being synchronized. Following the transition period, all output clocks begin toggling at the new divider values simultaneously. The transition from the old divider value to the new divider value takes no more than 100 ns. Because the output divider transition takes a period of time to change, the PLL_CR may not be written back-to-back without waiting 100 ns between writes.

7.3.2 PLL Frequency Synthesis

The frequency synthesis of the PLL is defined by the feedback divider (P-divider) and reference divider (Q-divider) in Equation 7-7:

$$f_{VCO} = f_{REF} \times \frac{P}{Q} = f_{REF} \times \left(\frac{PLL\_CR[FBKDIV] + 1}{2^{PLL\_CR[REFDIV]}}\right) \qquad \textbf{\textit{Eqn. 7-7}}$$

The allowable range of values for FBKDIV is 0–63 and REFDIV is 0–1. The following constraints must be met:

- f_{VCO} must be in the range 240–480 MHz.
- f_{REF} must be in the range 14–40 MHz.
- $\frac{f_{REF}}{2^{PLL\_CR[REFDIV]}}$ must be in the range 14–40 MHz.

The PLL_CR can be modified during normal operation or while the device is in limp mode. See Section 7.2.1, "PLL Control Register (PLL_CR)", for further information about FBKDIV and REFDIV.

7.3.3 Lock Conditions

When PLL_CR[LOLEN] is set, the lock-detect logic monitors the PFD reference clock and the feedback clock to determine when frequency lock has been achieved. Phase lock is inferred by the frequency relationship, but is not guaranteed. The PLL lock status is reflected in PLL_SR[LOCK].

The lock-detect function uses two counters that are clocked by the divided reference and feedback clocks respectively. When the reference counter has counted N cycles, the feedback counter's count is compared. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then, if the two counters still match, the lock criteria is relaxed by one count, and the system is notified that the PLL has achieved frequency lock.

After lock is detected, the lock circuitry continues to monitor the reference and feedback clocks using the alternate count-and-compare process. If the counters do not match at any comparison time, then PLL_SR[LOCK] is cleared to indicate that the PLL has lost lock. At this point, the lock criteria is tightened and the lock-detect process is repeated. The alternate count sequences prevent false lock detects due to frequency aliasing while the PLL tries to lock. Alternating between a tight and relaxed lock criteria prevents the lock-detect function from randomly toggling between locked and not locked status due to phase sensitivities.

When PLL_CR[LOLEN] is cleared, PLL_SR[LOCK] is set following a count of 2000 reference clock cycles, indicating sufficient time has elapsed for the PLL to reach a locked state. PLL_SR[LOCK] is cleared by asserting system reset.

In limp mode, since the PLL is disabled, there is no PLL lock status.

7.3.4 Loss-of-Lock Detection

When the PLL achieves lock following a system reset, PLL_SR[LOCK, LOCKS] are set. When the PLL loses lock, both bits are cleared. While the PLL is in an unlocked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to re-lock. Therefore, during the re-locking process, the system clock frequency is not well defined and may exceed the maximum system frequency. This violates the system clock timing specifications. Due to this condition, we recommend using the loss-of-lock reset functionality as described in Section 7.3.4.1, “Loss-of-Lock Reset Request”.

When the PLL has re-locked, the PLL does not update the PLL_SR[LOCKS] bit indicating lock has been lost since a system reset. This bit is sticky and must be cleared by writing a one to it before the PLL writes the register again.

7.3.4.1 Loss-of-Lock Reset Request

The PLL can assert reset when a loss-of-lock condition occurs by programming the PLL_SR[LOLRE] bit. Because the PLL_SR[LOCK, LOCKS] bits are cleared after reset, the reset status register (RSR) must be read to determine a loss-of-lock condition occurred. See Section 10.3.2, “Reset Status Register (RSR)”, for more information on the RSR register. To exit reset in PLL mode, the reference must be present and the PLL must acquire lock.

7.3.4.2 Loss-of-Lock Interrupt Request

The PLL can request an interrupt when a loss-of-lock condition occurs by programming the PLL_SR[LOLIRQ] bit. If this bit is set and a loss of lock is detected, PLL_SR[LOLF] is set, requesting an interrupt to the processor. The LOLF bit is sticky and remains set until it is cleared by a system reset or by writing a one to it. LOLIRQ provides information to the lock-detect logic to let it know if an interrupt should be generated upon loss of lock.

7.3.5 Loss-of-Clock Detection

When PLL_CR[LOCEN] is set, the loss-of-reference logic monitors the PFD reference clock for a loss-of-clock condition. For every rising edge of the PFD reference clock, a logic 1 is written to a register, which is reset every eight feedback clocks. This register is read just before being reset. If found to be a logic 0, a loss-of-clock condition is declared. The reset cycle of eight feedback clocks was chosen to prevent minor phase variations between the PFD reference and feedback clocks triggering loss of clock. When loss of reference is detected, PLL_SR[LOC] is set. After the reference clock is restored, PLL_SR[LOC] is cleared.

When PLL_CR[LOCEN] is cleared, the loss-of-reference logic is disabled.

7.3.5.1 Loss-of-Clock Reset Request

When you set PLL_SR[LOCRE, LOCEN], the PLL requests a reset when a loss-of-clock condition occurs. Since the PLL_SR[LOC] bit is cleared after reset, the reset controller’s RSR register must be read to determine that a loss-of-clock condition occurred. See Section 10.3.2, “Reset Status Register (RSR)”, for more information on the RSR register. To exit reset in PLL mode, the reference must be present and the PLL must acquire lock. PLL_SR[LOCRE] is ignored when PLL_SR[LOCEN] is cleared.

7.3.5.2 Loss-of-Clock Interrupt Request

When you set PLL_CR[LOCIRQ, LOCEN], the PLL requests an interrupt when a loss-of-clock condition occurs. If this bit is is set and loss of lock is detected, the PLL sets PLL_SR[LOCF] and asserts an interrupt request to the processor. The LOCF bit is sticky and remains set until it is cleared by writing a one to it or by asserting a system reset. PLL_SR[LOCIRQ] is ignored when PLL_SR[LOCEN] is cleared.

7.3.6 System Clock Modes

After reset, the PLL is placed in reset and the system runs in limp mode. You must clear MISCCR[LIMP] after reset to enable the PLL. See Section 9.3.4, “Miscellaneous Control Register (MISCCR)”, for details of this register.

Table 7-6 shows the clock-out to clock-in frequency relationships for the possible system clock modes.

Table 7-6. System Clock And Reference Clock Relationship

Clock Mode	PLL option[1]	Reference
Normal PLL mode	$f_{SYS} = \dfrac{f_{VCO}}{\text{PLL\_DR[OUTDIV1]} + 1}$	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, f_{VCO} = VCO frequency
MISCCR[LPDIV] ranges from 0 to 15

7.3.7 Clock Operation During Reset

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

7.3.7.1 Power-On Reset (POR)

After VDD_PLL and the input clock are within specification, the PLL is held in reset for at least ten input clock cycles to initialize the PLL. The reset configuration signals are used to select the multiply factor of the PLL and the reset state of the PLL registers. While in reset, the PLL input clock is output to the device. After $\overline{\text{RESET}}$ de-asserts, PLL output clocks generate; however, until the PLL_SR[LOCK] bit is set, the PLL output clock frequencies are not stable and within specification. When this bit is set, the PLL is in frequency lock.

7.3.7.2 External Reset

When $\overline{\text{RESET}}$ asserts, the PLL input clock outputs to the device, and the PLL does not begin acquiring lock until $\overline{\text{RESET}}$ is negated. The PLL_SR[LOCK] bit is cleared and remains cleared while the PLL is acquiring lock.

CAUTION

When running in an unlocked state, the clocks the PLL generate are not guaranteed to be stable and may exceed the maximum specified frequency.

Chapter 8
Power Management

8.1 Introduction

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

8.1.1 Features

These features support low-power operation:

- Five operation modes: run, wait, doze, stop, and standby
- 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
- Ability to continue running the real-time clock when rest of the device is powered-off
- Ability to maintain internal SRAM when rest of the device is powered-off

8.2 Memory Map/Register Definition

The power management programming model consists of registers from the SCM and CCM memory space.

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_0030	Peripheral Power Management High Register 0 (PPMHR0)	32	R/W	0xFFFC_00D0	8.2.4/8-4
0xFC04_0034	Peripheral Power Management Low Register 0 (PPMLR0)	32	R/W	0x0000_0300	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	9.3.4/9-5

[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 (WCR)

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

NOTE

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

Sequence of operations generally needed to enable this functionality:

1. The WCR register is programmed, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. Once the processor stops execution, it asserts a specific processor status (PST) encoding. Issuing the STOP instruction when the WCR[ENBWCR] is set causes the SCM to enter the mode specified in WCR[LPMD].
3. The low power mode control logic processes the entry into a low power mode, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. After an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.
6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

Address: 0xFC04_0013 (WCR) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	ENBWCR	0	LPMD		0	PRILVL		
W		▨			▨			
Reset:	0	0	0	0	0	0	0	0

Figure 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 WCR[LPMD]. 0 Low-power mode entry is disabled 1 Low-power mode entry is enabled.
6	Reserved, must be cleared.

Table 8-2. WCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5–4
LPMD</td><td>Low-power mode select. Used to select the low-power mode the chip enters once the ColdFire core executes the STOP instruction. To take effect, write these bits prior to instruction execution. The LPMD bits are readable and writable in all modes.
00 Run
01 Doze
10 Wait
11 Stop
<b>Note:</b> If WCR[LPMD] is cleared, the device stops executing code upon a STOP instruction. However, no clocks disable.</td></tr>
<tr><td>3</td><td>Reserved, must be cleared.</td></tr>
<tr><td>2–0
PRILVL</td><td>Exit low-power mode interrupt priority level. This field defines the interrupt priority level to exit the low-power mode:
<table>
<tr><th>PRILVL</th><th>Interrupt Level Needed to Exit Low-Power Mode</th></tr>
<tr><td>000</td><td>Any interrupt request exits low-power mode</td></tr>
<tr><td>001</td><td>Interrupt request levels [2-7] exit low-power mode</td></tr>
<tr><td>010</td><td>Interrupt request levels [3-7] exit low-power mode</td></tr>
<tr><td>011</td><td>Interrupt request levels [4-7] exit low-power mode</td></tr>
<tr><td>100</td><td>Interrupt request levels [5-7] exit low-power mode</td></tr>
<tr><td>101</td><td>Interrupt request levels [6-7] exit low-power mode</td></tr>
<tr><td>11<i>x</i></td><td>Interrupt request level [7] exits low-power mode</td></tr>
</table>
</td></tr>
</table>

8.2.2 Peripheral Power Management Set Register (PPMSR0)

The PPMSR register provides a simple mechanism to set a given bit in the PPM{H,L}R registers to disable the clock for a given peripheral module without needing to perform a read-modify write on the PPMR. The data value on a register write causes the corresponding bit in the PPM{H,L}R to be set. The SAMCD bit provides a global set function forcing the entire contents of the PPMR to set, disabling all peripheral module clocks. Reads of these registers return all zeroes.

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

	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 Register (PPMSR0)

Table 8-3. PPMSR0 Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SAMCD	Set all module clock disables. 0 Set only those bits specified in the SMCD field 1 Set all bits in PPMRH and PPMRL, disabling all peripheral clocks
5–0 SMCD	Set module clock disable. Set the corresponding bit in PPM{H,L}R, disabling the peripheral clock.

8.2.3 Peripheral Power Management Clear Register (PPMCR0)

The PPMCR register provides a simple mechanism to clear a given bit in the PPMHR & PPMLR registers, enabling the clock for a given peripheral module without needing to perform a read-modify write on the PPMR. The data value on a register write causes the corresponding bit in the PPM{H,L}R to be clear. A value of 64 to 127 (setting the CAMCD bit) provides a global clear function, forcing the entire PPMR contents to clear, enabling all peripheral module clocks. Reads of these registers return all zeroes.

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

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	▨	CAMCD	CMCD					
Reset:	0	0	0	0	0	0	0	0

Figure 8-3. Peripheral Power Management Clear Register (PPMCR0)

Table 8-4. PPMCR0 Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAMCD	Clear all module clock disables. 0 Clear only those bits specified in the CMCD field 1 Clear all bits in PPMRH and PPMRL, enabling all peripheral clocks
5–0 CMCD	Clear module clock disable. Clear the corresponding bit in PPMR{H,L}, enabling the peripheral clock.

8.2.4 Peripheral Power Management Registers (PPMHR0 & PPMLR0)

The PPMR registers provide a bit map for controlling the generation of the peripheral clocks for each decoded address space. Recall each peripheral module is mapped into 16 kByte slots within the memory map. The PPMR registers provide a unique control bit for each address space that defines whether the module clock for the given space is enabled or disabled.

Because the operation of the crossbar switch and the system control module (SCM) are fundamental to the operation of the device, the clocks for these modules cannot be disabled.

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

Address: 0xFC04_0030 (PPMHR0) Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	1	1	1	1	CD51	CD50	CD49	CD48
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	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	CD39	1	CD37	CD36	CD35	CD34	CD33	CD32
W																
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 8-4. Peripheral Power Management High Register (PPMHR0)

Table 8-5. PPMHR0[CD*n*] Assignments

Slot Number	CD*n*	Peripheral
32	CD32	PIT 0
33	CD33	PIT 1
34	CD34	PIT 2
35	CD35	PIT 3
36	CD36	Edge port 0
37	CD37	Edge port 1
39	CD39	Codec
40	CD40	CCM, Reset Controller, Power Management
41	CD41	Pin Multiplexing and Control (GPIO)
42	CD42	Real time clock (RTC)
43	CD43	SIM (Smartcard interface)
44	CD44	USB On-the-Go
45	CD45	USB host
46	CD46	SDRAM Controller
47	CD47	SSI
48	CD48	PLL
49	CD49	Random number generator (RNG)
50	CD50	IIM
51	CD51	eSDHC

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	CD13	CD12	0	0	0	0	0	0	CD5	0	0	CD2	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-5. Peripheral Power Management Low Registers (PPMLR0)

Table 8-6. PPMLR0[CD*n*] Assignments

Slot Number	CD*n*	Peripheral
2	CD2	FlexBus
5	CD5	MPU
12	CD12	FEC0
13	CD13	FEC1
17	CD17	eDMA Controller
18	CD18	Interrupt Controller 0
19	CD19	Interrupt Controller 1
21	CD21	IACK
22	CD22	I^2C
23	CD23	DSPI
24	CD24	UART0
25	CD25	UART1
26	CD26	UART2
28	CD28	DMA Timer 0
29	CD29	DMA Timer 1
30	CD30	DMA Timer 2
31	CD31	DMA Timer 3

Table 8-7. 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

Take extreme caution when setting PPMR[CD40] to disable clocking of the CCM, reset controller, and power management modules. This may disable logic to reset the chip and disable the external bus monitor or other logic contained within these blocks.

8.2.5 Low-Power Control Register (LPCR)

The LPCR register controls chip and module operation during low-power modes.

Address: 0xFC0A_0007 (LPCR)

Access: Supervisor read/write

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

Figure 8-6. Low-Power Control Register (LPCR)

Table 8-8. LPCR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 FWKUP	Fast wake-up. Determines whether the system clocks are enabled upon wake-up from stop mode. This bit must be written before execution of the STOP instruction for it to take effect. 0 System clocks enabled only when PLL is locked or operating normally 1 System clocks enabled upon wake-up from stop mode, regardless of PLL lock status **Note:** Setting this bit is potentially dangerous and unreliable. The system may behave unpredictably when using an unlocked clock, because the clock frequency could overshoot the maximum frequency of the device. **Note:** If FWKUP is set before entering stop mode, it should not be cleared upon wake-up from stop mode until after the PLL has actually acquired lock. Lock status may be obtained by reading MISCCR[PLLLOCK]. Because the PLL never locks in limp mode, FWKUP is ineffective. The system clocks are always enabled upon wake-up from stop mode, regardless of the value of FWKUP.
4–3 STPMD	FB_CLK stop mode bits. Controls the operation of the clocks, PLL, and oscillator in stop mode:
2–0	Reserved, must be cleared.

STPMD	System Clocks	FB_CLK	PLL	Oscillator
00	Disabled	Enabled	Enabled	Enabled
01	Disabled	Disabled	Enabled	Enabled
10	Disabled	Disabled	Disabled	Enabled
11	Disabled	Disabled	Disabled	Disabled

8.3 Functional Description

This section discusses the functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes.

8.3.1 Peripheral Shut Down

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

8.3.2 Limp mode

The device boots into a low-frequency limp mode, in which the PLL is bypassed and the device runs from a factor of the input clock (EXTAL). In this mode, EXTAL feeds a counter that divides the input clock by 2^n, where *n* is the value of the programmable counter field, CDR[LPDIV]. The programmed value of the divider may be changed without glitches or otherwise negative affects to the system. While in this mode, the PLL is placed in bypass mode to reduce overall system-power consumption.

Limp mode may also be entered and exited by writing to MISCCR[LIMP].

While in this mode, the core and primary bus clock maintains a 3:1 ratio. Because they will not function at speeds as low as the minimum input clock frequency, the SDRAM controller and FECs are not functional in limp mode. Also, the USB host and OTG controllers must source their system clocks through alternate, off-chip sources.

8.3.3 Standby Mode

The real-time clock and SRAM each contain a standby voltage pin (VSTBY_RTC and VSTBY_SRAM, respectively), allowing them to remain active when the processor is powered off.

8.3.4 Low-Power Modes

The system enters a low-power mode by executing a STOP instruction. The low-power mode the device actually enters (stop, wait, or doze) depends on the setting of the WCR[LPMD] bits. Entry into any of these modes idles the CPU with no cycles active, powers down the system, and stops all internal clocks appropriately. During stop mode, the system clock is stopped low.

A wake-up event is required to exit a low-power mode and return to run mode. Wake-up events consist of any of these conditions:

- Any type of reset
- Any valid, enabled interrupt request

Exiting from low-power mode via an interrupt request requires:

- An interrupt request whose priority is higher than the value programmed in the WCR[PRILVL].

- An interrupt request whose priority is higher than the value programmed in the interrupt priority mask (I) field of the core's status register.
- An interrupt request from a source not masked in the interrupt controller's interrupt mask register.
- An interrupt request which has been enabled at the module of the interrupt's origin.

8.3.4.1 Run Mode

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

8.3.4.2 Wait Mode

Wait mode is intended to stop only the CPU and memory clocks until a wake-up event is detected. In this mode, peripherals may be programmed to continue operating and can generate interrupts, causing the CPU to exit from wait mode.

8.3.4.3 Doze Mode

Doze mode affects the processor in the same manner as wait mode, except that some peripherals define individual operational characteristics in doze mode. Peripherals continuing to run and having the capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Stopped peripherals restart operation on exit from doze mode, as defined for each peripheral.

8.3.4.4 Stop Mode

Stop mode affects the processor the same as the wait and doze modes, except that all clocks to the system are stopped and the peripherals cease operation.

Stop mode must be entered in a controlled manner to ensure that any current operation is properly terminated. When exiting stop mode, most peripherals retain their pre-stop status and resume operation.

NOTE

> Entering stop mode disables the SDRAMC, including the refresh counter. If SDRAM is used, code is required to ensure proper entry and exit from stop mode. See Chapter 19, "SDRAM Controller (SDRAMC)," for more information.

8.3.5 Peripheral Behavior in Low-Power Modes

The following subsections specify the operation of each module while in and when exiting low-power modes. In wait and doze modes, the clocks to the CPU and SRAM stop 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 & PPMLR0)"). In stop mode, all clocks to the system stop.

8.3.5.1 ColdFire Core

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

8.3.5.2 Internal SRAM

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

8.3.5.3 Clock Module

There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wake-up recovery time. The PLL can be disabled in stop mode, but requires a wake-up period before it can relock. The oscillator can also be disabled during stop mode, but requires a wake-up period to restart.

When the PLL is enabled in stop mode (LPCR[STPMD] = 00), the external FB_CLK signal can support systems using FB_CLK as the clock source. See Section 8.2.5, "Low-Power Control Register (LPCR)," for more information about operating the PLL in stop mode.

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

8.3.5.4 Chip Configuration Module

The chip configuration module is unaffected by entry into a low-power mode. However, CCM register access is disabled. If reset occurs exiting the low-power mode, chip configuration may execute if configured to do so.

8.3.5.5 Reset Controller

A power-on reset (POR) always causes a chip to reset and exit from any low-power mode.

In wait and doze modes, asserting the external $\overline{\text{RESET}}$ pin for at least four clocks causes an external reset that resets the chip and exits any low-power modes.

In stop mode, the $\overline{\text{RESET}}$ pin synchronization is disabled and asserting the external $\overline{\text{RESET}}$ pin asynchronously generates an internal reset and exits any low-power modes. Registers lose their current values and must be reconfigured from their reset state if needed.

If the core watchdog timer is still enabled during wait or doze modes, a watchdog timer timeout may generate a reset to exit these low-power modes.

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

8.3.5.6 System Control Module (SCM)

The SCM’s core watchdog timer can bring the device out of all low-power modes except stop mode. In stop mode, all clocks stop and the core watchdog timer does not operate.

When enabled, the core watchdog can bring the device out of low-power mode in one of two ways depending on the setting of the CWCR[CWRI] field.

- A core watchdog timeout may reset the device.
- Upon a watchdog timeout, a watchdog interrupt can bring the device out of low-power mode. This system setup must meet the conditions specified in Section 8.3.4, “Low-Power Modes”.

8.3.5.7 Crossbar Switch

The crossbar switch is disabled in stop mode and is enabled in all other low-power modes.

8.3.5.8 GPIO Ports

The GPIO ports are unaffected by entry into a low-power mode. In stop mode, the register space associated with the GPIO ports is not accessible. 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 revert to their default direction settings.

8.3.5.9 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 stop.

An interrupt request causes the processor to exit a low-power mode only if that interrupt’s priority level is at or above the level programmed in the interrupt priority mask field of the CPU’s status register (SR) and above the level programmed in the WCR[PRILVL]. The interrupt must also be enabled in the interrupt controller’s interrupt mask register, as well as at the module from which the interrupt request originates.

8.3.5.10 Edge Ports

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

In stop mode, no system clock is available to perform the edge-detect function. Therefore, only the level-detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

8.3.5.11 eDMA Controller

In wait and doze modes, the eDMA controller can bring the device out of a low-power mode by generating an interrupt upon completion of a transfer or upon an error condition. The completion of transfer interrupt generates when DMA interrupts are enabled by the setting of a EDMA_INTR[INT*n*], and an interrupt is generated when TCD*n*[DONE] is set. The interrupt upon error condition occurs when EDMA_EEIR[EEI*n*] is set, and an interrupt generates when any of the EDMA_ESR bits become set.

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

8.3.5.12 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.

8.3.5.13 SDRAM Controller (SDRAMC)

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

To prevent data loss, SDRAMC should be placed in self-refresh mode by clearing SDCR[CKE] and setting SDCR[REF_EN]. The SDRAM self-refresh mode allows the SDRAM to enter a low-power state where internal refresh operations maintain the integrity of the SDRAM data.

When stop mode is exited, setting SDCR[CKE] causes the SDRAM controller to exit the self-refresh mode and allows bus cycles to the SDRAM to resume.

NOTE

The SDRAM is inaccessible in the self-refresh mode. Therefore, if stop mode is used, the vector table and any interrupt handlers that could wake the processor should not be stored in or attempt to access SDRAM.

8.3.5.14 Fast Ethernet Controllers (FEC)

In wait and doze modes, the FEC is unaffected and may generate an interrupt to exit these low-power modes. In stop mode, the FEC stops immediately and freezes operation, register values, state machines, and external pins. The FEC clocks also shut down in this mode. Exiting stop mode returns the FEC to operation from the state prior to stop mode entry.

8.3.5.15 eSDHC

When the clocks to the eSDHC are disabled or when the system is in a low power mode, the eSDHC can generate the following three wake-up interrupts:

- Card removal interrupt
- Card insertion interrupt
- SDIO card interrupt

The eSDHC offers a power management feature. By clearing the clock-enabled bits in the clock control register, the clocks are gated low to the eSDHC. For maximum power saving, the user can disable all the clocks to eSDHC when there is no operation in progress.

8.3.5.16 Cryptography Acceleration Unit (CAU)

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

8.3.5.17 Subscriber Interface Module (SIM)

The SIM is enabled in wait mode. In doze mode SIM operation is configurable with the SIM_RCR[DOZE] bit. If this bit is set, the SIM gates its clocks when the transmit FIFO is empty. If cleared, doze mode has no effect on the SIM.

In stop mode the SIM can be disabled completely or partially disabled. If SIM_RCR[STOP] is set, only the SIM card baud clock runs. If cleared, all clocks are disabled.

8.3.5.18 Codec

Since the clocks to the codec are enabled during doze and wait modes, the codec functions during these modes. Since the bus clock is disabled during stop mode, the codec is not able to transmit/receive data during this mode. Entry into stop mode is delayed until the codec completes the current processing from the ADC path.

NOTE

Since analog part of the codec still receives the reference clock from the oscillator, the clock needs to be gated during stop mode to turn off the analog part.

8.3.5.19 IIM/Polyfuse (Fusebox)

Since the clocks to the IIM and fusebox are enabled during wait or doze mode, they continue to function during these modes.

Since the clocks are disabled during stop mode, the IIM cannot access the IIM/Fusebox registers. Hardware-visible nets that go across the SoC from the IIM/fusebox are still driven and are not affected by stop mode. These include keys, bandgap trimming, amplifiers controls, etc. that are being directly driven from the IIM output signals.

8.3.5.20 Random Number Generator (RNG)

Since the clocks to the RNG are enabled during wait and doze modes, the RNG continues to function during these modes.

The RNG does not function in stop mode since the clocks are disabled. Entry into stop mode is delayed until it completes the current seed generation.

8.3.5.21 USB On-the-Go Module

In wait mode the clocks to the USB OTG module continue to run. In doze mode, the processor stops the 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.

In stop mode, the processor stops the clock to the USB OTG module. In this state, the 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.

The USB block contains an automatic low power mode in which the module enters suspend mode after a 6.0 ms minimum period of inactivity. When the module receives a wake-up from the USB host, the transceiver is re-enabled for normal USB operations.

8.3.5.22 USB Host Module

In wait mode the clocks to the USB host module continue to run. In doze mode, the processor stops the clocks to the USB host module, but the 60 Mhz transceiver clock remains active. In doze mode, detection of resume signaling initiates a restart of the module clocks.

In stop mode, the processor stops the clock to the USB host module. In this state, the 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.

8.3.5.23 Real Time Clock

In stop mode, the external clock driving RTC_EXTAL/RTC_XTAL continues to clock the RTC module. Therefore the device is still able to update the RTC counters, alarms, etc. while in stop mode. An RTC interrupt/wake-up can be generated while in stop mode to wake up the device if the RTC alarms are triggered.

8.3.5.24 Programmable Interrupt Timers (PIT0–3)

In stop mode (or doze mode, if so programmed in the PCSR*n* register), the programmable interrupt timer (PIT) ceases operation and freezes at the current value. When exiting these modes, the PIT resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the PIT may generate an interrupt to exit the low-power modes.

8.3.5.25 DMA Timers

In wait and doze modes, the DMA timers may generate an interrupt to exit a low-power mode. This interrupt can generate when the DMA timer is in input capture mode or reference compare mode.

In input capture mode, if DTMR[CE] is non-zero and DTXMR[DMAEN] is cleared, an interrupt issues upon a captured input. In reference compare mode, if DTMR[ORRI] is set and DTXMR[DMAEN] is cleared, an interrupt issues when the timer counter reaches the reference value.

Entering stop mode disables DMA timer operation. Upon exiting stop mode, the timer resumes operation unless stop mode was exited by reset.

8.3.5.26 DMA Serial Peripheral Interface (DSPI)

In wait mode, the DSPI module is unaffected and may generate an interrupt to exit this low-power mode.

In stop and doze modes (if doze is enabled in DSPI), the DSPI first completes transfer of the current frame. It then freezes operation, register values, state machines, and external pins. During this mode, the DSPI clocks are shut down. Exiting stop mode returns the DSPI to operation from the state prior to stop mode entry.

8.3.5.27 UART Modules (UART0–2)

In wait and doze modes, the UARTs are unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the UARTs stop immediately and freeze their operation, register values, state machines, and external pins. During this mode, the UART clocks shut down. Exiting stop mode returns the UARTs to the operation of the state prior to stop-mode entry.

8.3.5.28 I^{2}C Module

When the I^{2}C module is enabled by setting I2CR[IEN] and the device is not in stop mode, the I^{2}C module is operable and may generate an interrupt to bring the device out of a low-power mode. For an interrupt to occur, I2CR[IIE] must be set to enable interrupts, and setting I2SR[IIF] generates the interrupt signal to the CPU and interrupt controller. Setting 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.

8.3.5.29 BDM

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

8.3.5.30 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 processor to exit any low-power mode. Toggling TCLK during any low-power mode increases the system current consumption.

8.3.6 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 8-9. The status of each peripheral during a given mode refers to the condition the peripheral automatically assumes when the STOP instruction is executed and the WCR[LPMD] field is set for the particular low-power mode. Individual peripherals may be disabled by programming its dedicated control bits. The wake-up procedure field refers to the ability of an interrupt or reset by that peripheral to force the CPU into run mode.

Table 8-9. CPU and Peripherals in Low-Power Modes

Module	Peripheral Status[1] / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
ColdFire Core	Stopped	N/A	Stopped	N/A	Stopped	N/A
CAU	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	Reset
GPIO	Enabled	N/A	Enabled	N/A	Stopped	N/A
Interrupt controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
Edge ports	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
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 Controllers	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
USB OTG	Enabled	Interrupt	Disabled	Interrupt	Stopped	N/A
USB Host	Enabled	Interrupt	Disabled	Interrupt	Stopped	N/A
eSDHC	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
IIM	Enabled	N/A	Enabled	N/A	Stopped	N/A
Codec	Enabled	N/A	Enabled	N/A	Stopped	N/A
SIM	Enabled	Interrupt	Program	Interrupt	Stopped	Interrupt
SSI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
Real Time Clock	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
Programmable Interrupt Timers	Enabled	Interrupt	Program	Interrupt	Stopped	N/A
DMA Timers	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
DSPI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
UARTs	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A

Table 8-9. CPU and Peripherals in Low-Power Modes (continued)

Module	Peripheral Status[1] / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
I^{2}C Module	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
RNG	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
JTAG[2]	Enabled	N/A	Enabled	N/A	Enabled	N/A
BDM[3]	Enabled	Yes	Enabled	Yes	Enabled	Yes

[1] "Program" indicates that the peripheral function during the low-power mode is dependent on programmable bits in the peripheral register map.

[2] The JTAG logic is clocked by a separate TCLK clock.

[3] Entering halt mode via the BDM port exits any lower-power mode. Upon exit from halt mode, the previous low-power mode will be re-entered, and changes made in halt mode will remain in effect.

Chapter 9
Chip Configuration Module (CCM)

9.1 Introduction

The chip-configuration module (CCM) configures the device to operate in one of multiple functional modes.

9.1.1 Block Diagram

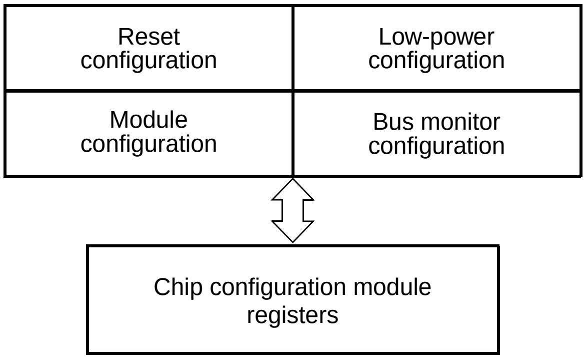


Figure 9-1. Chip-Configuration Module Block Diagram

9.1.2 Features

The CCM performs these operations:

- Configures device based on chosen reset configuration options
- 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 FB_R/$\overline{\text{W}}$, $\overline{\text{FB\_TS}}$, $\overline{\text{FB\_TA}}$, $\overline{\text{FB\_OE}}$, and $\overline{\text{FB\_BE/BWE}}$[3:0]. Up to four 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
BOOTMOD[1:0]	Reset configuration select	—
FB_A[21:17]	Reset configuration override pins	—

9.2.1 BOOTMOD[1:0]

If the BOOTMOD[1:0] signals determine the boot performed at reset. See the table below for BOOTMOD[1:0] usage.

Table 9-2. BOOTMOD[1:0] Values

BOOTMOD[1:0]	Meaning
00	Boot from FlexBus with defaults, unified SDR bus/FlexBus
01	Boot from FlexBus with defaults, split DDR bus/FlexBus
10	Boot from FlexBus and override defaults via address bus (FB_A[21:17])
11	Boot from FlexBus and override defaults via address bus (FB_A[21:17])

9.2.2 FB_A[21:17] (Reset Configuration Override)

If the external BOOTMOD[1:0] pins are driven to 10 during reset, the states of the FB_A[21:17] pins during reset determine Flexbus, SDRAM, and PLL configurations after reset.

NOTE

The logic levels for reset configuration on FB_A[21:17] must be actively driven when BOOTMOD equals 10. FB_A[23:22,16:0] should 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 9-3. CCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers[1]					
0xFC0A_0004	Chip Configuration Register (CCR)	16	R	See Section	9.3.1/9-3
0xFC0A_0008	Reset Configuration Register (RCON)	16	R	0x0003	9.3.2/9-4
0xFC0A_000A	Chip Identification Register (CIR)	16	R	See Section	9.3.3/9-5

Table 9-3. CCM Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0010	Miscellaneous Control Register (MISCCR)	16	R/W	0x583D	9.3.4/9-5
0xFC0A_0012	Clock Divider Register (CDR)	16	R/W	0x0001	9.3.5/9-7
0xFC0A_0014	USB On-the-Go Controller Status Register (UOCSR)	16	R/W	0x3010	9.3.6/9-7
0xFC0A_0016	USB Host Controller Status Register (UHCSR)	16	R/W	0x0000	9.3.7/9-9
0xFC0A_001A	Codec Control Register	16	R/W	0x0000	9.3.8/9-10
0xFC0A_001C	Miscellaneous Control Register 2 (MISCCR2)	16	R/W	0x0001	9.3.9/9-10

[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 it has no effect. At reset, the CCR reflects the chosen operation of certain device functions. These functions may be set to the defaults defined by the RCON values or overridden during reset configuration using the external BOOTMOD[1:0] and the FB_A[21:17] 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	0	0	0	0	CSC	BOOT PS	LOAD	OSC MODE	SDR MODE	1
W																
Reset	0	0	0	0	0	0	0	0	0	0	See Note					1

Note: Reset value depends upon chosen reset configuration. Default reset value (BOOTMOD = 00) is the value of RCON.

Figure 9-2. Chip Configuration Register (CCR)

Table 9-4. CCR Field Descriptions

Field	Description
15–6	Reserved, must be cleared.
5 CSC	Chip select configuration. Relects the chosen configuration for the FB_A[23:22] pins. 0 $\overline{\text{FB\_CS}}$[3:2] 1 FB_A[23:22]

Table 9-4. CCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>4
BOOTPS</td><td>Boot port size. Along with SDRMODE, determines the boot port size.
<table>
<tr><th>SDRMODE</th><th>BOOTPS</th><th>Description</th></tr>
<tr><td>0</td><td>0</td><td>16-bit split data bus (DDR)</td></tr>
<tr><td>0</td><td>1</td><td>8-bit split data bus (DDR)</td></tr>
<tr><td>1</td><td>0</td><td>16-bit unified data bus (SDR)</td></tr>
<tr><td>1</td><td>1</td><td>32-bit unified data bus (SDR)</td></tr>
</table>
</td></tr>
<tr><td>3
LOAD</td><td>Pad driver load bit. Reflects the chosen pad driver strength for those pins with drive strength control.
0 Low drive strength (20pF)
1 High drive strength (50pF)</td></tr>
<tr><td>2
OSCMODE</td><td>Oscillator clock mode.
0 Crystal oscillator mode
1 Oscillator bypass mode</td></tr>
<tr><td>1
SDRMODE</td><td>SDRAM/FlexBus mode.
0 DDR mode, split external data bus
1 SDR mode, unified external data bus</td></tr>
<tr><td>0</td><td>Reserved, must be set.</td></tr>
</table>

9.3.2 Reset Configuration Register (RCON)

At reset, the RCON register determines the default operation of certain chip functions. All default functions defined by the RCON values can be overridden only during reset configuration (see Section 9.4.1, “Reset Configuration”) if the external BOOTMOD[1:0] pins are driven to 10. RCON is a read-only register and contains the same fields as the CCR register.

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	0	0	0	0	CSC	BOOT PS	LOAD	OSC MODE	SDR MODE	1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Figure 9-3. Reset Configuration Register (RCON)

9.3.3 Chip Identification Register (CIR)

CIR is a read-only register; writing to it has no effect.

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 9-5. CIR Field Descriptions

Field	Description
15–6 PIN	Part identification number. Contains a unique identification number for the device. 0x073 MCF53014 0x074 MCF53010 0x075 MCF53011 0x076 MCF53015 0x077 MCF53016 0x078 MCF53012 0x07C MCF53017 0x080 MCF53013
5–0 PRN	Part revision number. This number increases by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order.

9.3.4 Miscellaneous Control Register (MISCCR)

The MISCCR register provides various configuration options for limp mode, bus monitor, SSI/timer DMA mux, and SSI/USB clocks.

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	FECM	CDC SRC	PLL LOCK	LIMP	BME	BMT			0	0	TIM DMA	SSI SRC	USBH OC	USBO OC	USB PUE	USB SRC
W																
Reset	0	1	0	1	1	0	0	0	0	0	1	1	1	1	0	1

Figure 9-5. Miscellaneous Control Register (MISCCR)

Table 9-6. MISCCR Field Descriptions

Field	Description
15 FECM	FEC operating mode. Selects whether two external RMII PHYs or a single external MII PHY is used. 0 Single MII PHY 1 Two RMII PHYs
14 CDSRC	Codec clock source. 0 CODEC_ALTCLK drives the codec reference clock 1 Internal oscillator drives the codec reference clock

Table 9-6. MISCCR Field Descriptions (continued)

Field	Description
13 PLLLOCK	PLL lock status. Indicates the PLL is locked. 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 BME	Bus monitor external enable bit. Enables the bus monitor to operate during external FlexBus cycles 0 Bus monitor disabled on external FlexBus cycles 1 Bus monitor enabled on external FlexBus cycles
10–8 BMT	Bus monitor timing field. Selects the timeout period in FlexBus clock cycles for the bus monitor: Timeout period for external bus cycles equals $2^{(16\text{-}BMT)}$ FB_CLK cycles 000 65536 001 32768 010 16384 011 8192 100 4096 101 2048 110 1024 111 512
5 TIMDMA	Timer DMA mux selection. Selects between the timer DMA signals and SSI DMA signals as those signals are mapped to DMA channels 9-12. Refer to the Chapter 17, “Enhanced Direct Memory Access (eDMA),” for more details on the DMA controller. 0 SSI RX0, SSI RX1, SSI TX0, and SSI TX1 DMA signals mapped to DMA channels 9–12, respectively 1 Timer 0–3 DMA signals mapped to DMA channels 9–12, respectively
4 SSISRC	SSI clock source. Selects between the PLL and the external SSI_CLKIN pin as the source of the SSI baud clock. 0 SSI_CLKIN pin directly drives SSI baud clock 1 PLL drives SSI baud clock with fractionally divided CPU clock
3 USBHOC	USB host VBUS overcurrent sense polarity. Selects the polarity of the USB host controller’s VBUS over-current sense signal driven off-chip. 0 USBH_VBUS_OC is active low 1 USBH_VBUS_OC is active high
2 USBOC	USB On-the-Go VBUS over-current sense polarity. Selects the polarity of the USB OTG controller’s VBUS over-current sense signal driven off-chip. 0 USBO_VBUS_OC is active low 1 USBO_VBUS_OC is active high
1 USBPUE	USB transceiver pull-up enable. Enables the on-chip USB OTG controller to drive the internal transceiver pull-up. 0 Internal transceiver pull-up is disabled. The USB_PULLUP signal is used to trigger the external pull-up. 1 USB OTG drives the internal transceiver pull-up
0 USBSRC	USB clock source. Selects between the PLL and the external USB_CLKIN external pin as the clock source for the serial interface of the USB module. 0 USB_CLKIN pin drives USB serial interface clocks 1 PLL drives USB serial interface clocks

9.3.5 Clock-Divider Register (CDR)

The CDR register provides clock division factors for limp mode and the SSI master clock when the PLL is used to drive the SSI clock.

Address: 0xFC0A_0012 (CDR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	LPDIV				SSIDIV							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 9-6. Clock-Divider Register (CDR)

Table 9-7. CDR Field Descriptions

Field	Description
15–12	Reserved, must be cleared.
11–8 LPDIV	Low power clock divider. Specifies the divide value used to produce the system clocks during limp mode. A 3:1 ratio is maintained between the core and the internal bus.This field is used only when MISCCR[LIMP] is set. $$\text{System Clocks} = \frac{f_{\text{EXTAL}}}{2^{\text{LPDIV}}}$$ ***Eqn. 9-1*** **Note:** When LPDIV is cleared (divide by 1), the internal bus clock and FB_CLK does not have a 50/50 duty cycle.
7–0 SSIDIV	SSI oversampling clock divider. Specifies the divide value that produces the SSI oversampling clock. This field is used only when MISCCR[SSISRC] is set (PLL is the source). $$\text{SSI Baud Clock} = \frac{f_{\text{sys}}}{\text{SSIDIV}/2}$$ ***Eqn. 9-2*** **Note:** A value of 0 or 1 for SSIDIV represents a divide-by-65. SSIDIV must not be set to any value that sets the SSI oversampling clock frequency over the bus clock frequency ($f_{sys/2}$), because incorrect SSI operation could result.

9.3.6 USB On-the-Go Controller Status Register (UOCSR)

The UOCSR register controls and reflects various features of the USB OTG module. When any bit of this register generates an interrupt, that interrupt is cleared by reading the UOCSR register. The read-only bits of this register are set by the USB OTG module.

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	DPPD	DMPD	DRV VBUS	CRG VBUS	DCR VBUS	DPPU	AVLD	BVLD	VVLD	SEND	PWR FLT	WKUP	UOMIE	XPDE
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0

Figure 9-7. USB On-the-Go Controller Status Register (UOCSR)

Table 9-8. UOCSR Field Descriptions

Field	Description
15–14	Reserved, must be cleared.
13 DPPD	D+ 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D+ line is active. When set, asserts an interrupt if UOMIE is set. 0 Pull-down disabled 1 Pull-down enabled
12 DMPD	D- 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D- line is active. When set, asserts an interrupt if UOMIE is set. 0 Pull-down disabled 1 Pull-down enabled
11 DRVVBUS	Drive VBUS. 0 Disable the drive of 5 V power on VBUS 1 Enable the drive of 5 V power on VBUS
10 CRGVBUS	Charge VBUS. Indicates a charge resistor to pull-up VBUS is enabled. When set, asserts an interrupt if UOMIE is set. 0 Charge resistor to pull-up VBUS disabled 1 Charge resistor to pull-up VBUS enabled
9 DCRVBUS	Discharge VBUS. Indicates a discharge resistor to pull-down VBUS is enabled. When set, asserts an interrupt if UOMIE is set. 0 Discharge resistor to pull-down VBUS disabled 1 Discharge resistor to pull-down VBUS enabled
8 DPPU	D+ pull-up control. Indicates pull-up on D+ for FS-only applications is enabled. When set, asserts an interrupt if UOMIE is set. 0 D+ pull-up for FS-only applications disabled 1 D+ pull-up for FS-only applications enabled
7 AVLD	A-peripheral is valid. Indicates if the session for an A-peripheral is valid. 0 Session is not valid for an A-peripheral 1 Session is valid for an A-peripheral
6 BVLD	B-peripheral is valid. Indicates if the session for a B-peripheral is valid. 0 Session is not valid for a B-peripheral 1 Session is valid for a B-peripheral
5 VVLD	VBUS valid. Indicates if voltage on VBUS is at a valid level for operation. 0 Voltage level on VBUS is not valid for operation 1 Voltage level on VBUS is valid for operation
4 SEND	Session end. Indicates if voltage on VBUS has dropped below the session end threshold. 0 Voltage on VBUS has not dropped below the session end threshold 1 Voltage on VBUS has dropped below the session end threshold
3 PWRFLT	VBUS power fault. Indicates a power fault has occurred on VBUS (e.g. overcurrent). 0 No power fault has occurred 1 Power fault has occurred
2 WKUP	USB OTG controller wake-up event. Reflects if a wake-up event has occurred on the USB OTG controller bus. When set, asserts an interrupt if UOMIE is set. 0 No outstanding wake-up event 1 Wake-up event has occurred

Table 9-8. UOCSR Field Descriptions (continued)

Field	Description
1 UOMIE	USB OTG miscellaneous interrupt enable. Enables an interrupt to generate from any of the following UOCSR bits: DPPD, DMPD, CRGVBUS, DCRVBUS, DPPU, and WKUP 0 Interrupt sources are disabled 1 Interrupt sources are enabled
0 XPDE	On-chip transceiver pull-down enable. 0 50 kΩ pull-downs disabled on OTG D+ and D- pins of on-chip transceiver 1 On-chip 50 kΩ pull-downs enabled on OTG D+ and D- transceiver pins of on-chip transceiver

9.3.7 USB Host Controller Status Register (UHCSR)

The UHCSR register controls and reflects various features of the USB host module. When any bit of this register generates an interrupt, that interrupt is cleared by reading the UHCSR register. The read-only bits of this register are set by the USB host module.

Address: 0xFC0A_0016 (UHCSR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC		0	0	0	0	0	0	0	0	0	DRV VBUS	PWR FLT	WKUP	UHMIE	XPDE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-8. USB Host Controller Status Register (UHCSR)

Table 9-9. UHCSR Field Descriptions

Field	Description
15–14 PIC	Port indicator. Reflects the state of the USB host controller port indicator signals.
13–5	Reserved, must be cleared.
4 DRVVBUS	Drive VBUS. 0 Disable the drive of 5 V power on VBUS 1 Enable the drive of 5 V power on VBUS
3 PWRFLT	VBUS power fault. Indicates a power fault has occurred on VBUS (e.g. overcurrent). 0 No power fault has occurred 1 Power fault has occurred
2 WKUP	USB host controller wake-up event. Reflects if a wake-up event has occurred on the USB host controller bus. When set, asserts an interrupt if UHMIE is set. 0 No outstanding wake-up event 1 Wake-up event has occurred
1 UHMIE	USB host miscellaneous interrupt enable. Enables an interrupt to generate if WKUP sets. 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 host D+ and D- pins of on-chip transceiver 1 On-chip 50 kΩ pull-downs enabled on host D+ and D- transceiver pins of on-chip transceiver

9.3.8 Codec Control Register (CODCR)

CODCR enables the codec bandgap and regulator blocks.

Address: 0xFC0A_001A (CODCR) Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BGR EN	0	0	0	0	0	0	0	REG EN	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 9-9. Codec Control Register (CODCR)

Table 9-10. CODCR Field Descriptions

Field	Description
15 BGREN	Bandgap enable. Enables the analog bandgap reference block. 0 Bandgap reference not enabled 1 Bandgap reference enabled
14–8	Reserved, must be cleared.
7 REGEN	Regulator enable. Enables the analog regulator block. 0 Regulator not enabled 1 Regulator enabled
6–0	Reserved, must be cleared.

9.3.9 Miscellaneous Control Register 2 (MISCCR2)

MISCCR2 configures the PLL mode and fusebox programming delay.

Address: 0xFC0A_001C (MISCCR2) 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	PLLMODE			IGN LL	0	0	0	0	0	0	DPS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 9-10. Miscellaneous Control Register 2 (MISCCR2)

Table 9-11. MISCCR2 Field Descriptions

Field	Description
15–11	Reserved, must be cleared.
10–8 PLLMODE	PLL mode. 000 Function mode Else Reserved **Note:** This bit should never be altered.
7 IGNLL	FBCLK ignore PLL loss-of-lock. 0 FBCLK is gated during PLL loss-of-lock condition 1 FBCLK ignores the PLL loss-of-lock condition and is not gated

Table 9-11. MISCCR2 Field Descriptions (continued)

Field	Description
6–1	Reserved, must be cleared.
0 DPS	Fusebox delay program start. Delays fuse programming if IIM_FCR[DPC] is set. This feature is typically associated with power supply issues since the program operation consumes a lot of power. This bit may be tied to a low-power indicator, which would delay fuse programming until the processor is stopped (thus, extra current is available for the program operation). This bit is set after reset.

9.4 Functional Description

9.4.1 Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states. Table 9-12 shows the states of the external pins while in reset.

Table 9-12. Reset Configuration Pin States During Reset

Pin	Pin Function	I/O	Input State
BOOTMOD[1:0]	BOOTMOD function for all modes	I	Must be driven by external logic
FB_A[21:17]	Flexbus addressfunctions (BOOTMOD ≠ 1*x*)	I	N/A
	Reset configuration data functions (BOOTMOD = 1*x*)	I	Must be driven by external logic

9.4.1.1 Reset Configuration (BOOTMOD[1:0] = 00)

If the BOOTMOD pins are 00 during reset, the RCON register determines the chip configuration after reset, regardless of the states of the external address pins. The internal configuration signals are driven to levels specified by the RCON register’s reset state for default module configuration.

9.4.1.2 Reset Configuration (BOOTMOD[1:0] = 10 or 11)

If the BOOTMOD pins are 10 or 11 during reset, the chip configuration after reset is determined according to the levels driven onto the FB_A[21:17] pins. (See Table 9-13.) The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

NOTE

The logic levels for reset configuration on FB_A[21:17] must be actively driven when BOOTMOD is 10 or 11. The FB_A[23:22,16:0] pins must float or be pulled high.

Table 9-13. Parallel Configuration During Reset[1]

Pin(s) Affected	Default Configuration	Override Pins in Reset[2,3]	Function
FB_D[31:0]	See RCON[1]	**FB_A17**	**SDRAM/FlexBus Mode**
		0	DDR mode (split data bus)
		1	SDR mode (unified data bus)
(none)	See RCON[2]	**FB_A18**	**Oscillator Mode**
		0	Crystal oscillator mode
		1	Oscillator bypass mode
All output pins	See RCON[3]	**FB_A19**	**Load**
		0	Low drive strength (20 pF)
		1	High drive strength (50 pF)
FB_D[31:0]	See RCON[4]	**FB_A20**	**Boot Port Size**
		0	16-bit split or unified bus
		1	8-bit for split bus or 32-bit for unified bus
FB_A[23:22]/FB_CS[3:2]	See RCON[5]	**FB_A21**	**Chip Select Configuration**
		0	FB_A[23:22] = FB_CS[3:2]
		1	FB_A[23:22] = FB_A[23:22]

[1] Modifying the default configurations through the FB_A[21:17] pins is possible only if the external BOOTMOD[1:0] pins are 10 or 11 while RSTOUT is asserted.

[2] The FB_A[23:22,16:0] pins do not affect reset configuration.

[3] The external reset override circuitry drives the address bus pins with the override values while RSTOUT is asserted. It must stop driving the address bus pins within one FB_CLK cycle after 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 RSTOUT is negated.

9.4.2 Boot Configuration

During reset configuration, the FB_CS0 chip select pin is always configured to select an external boot device. The valid (V) bit in the CSMR0 register is ignored and FB_CS0 is enabled after reset. FB_CS0 is asserted for the initial boot fetch accessed from address 0x0000_0000 for the stack pointer and address 0x0000_0004 for the program counter (PC). It is assumed the reset vector loaded from address 0x0000_0004 causes the processor to start executing from external memory space decoded by FB_CS0.

9.4.3 Low Power Configuration

After reset, the device can be configured for operation during the low power modes using the low power control register (LPCR). For more information on this register, see Chapter 9, “Chip Configuration Module (CCM).”

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 these:

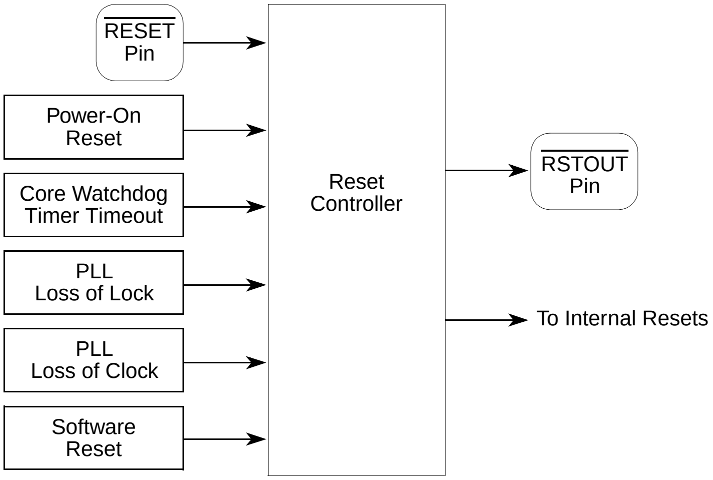


Figure 10-1. Reset Controller Block Diagram

10.1.2 Features

Module features include the following:

- Six sources of reset:
  - External
  - Power-on reset (POR)
  - Core watchdog timer
  - Phase locked-loop (PLL) loss of lock
  - Phase locked-loop (PLL) loss of reference clock
  - 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	I/O	Pull-up	Input Hysteresis	Input Synchronization
$\overline{\text{RESET}}$	I	Active	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 resets the device. It may take up to six FB_CLK edges after RESET assertion for RSTOUT to assert, due to an internal synchronizer on RESET. When $\overline{\text{RSTOUT}}$ is active, the user can drive override options on the data bus. See ,” 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 control 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 register descriptions.

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	10.3.1/10-3
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. 0 No software reset request 1 Software reset request
6 FRCRSTOUT	Allows software to assert or negate the external $\overline{\text{RSTOUT}}$ pin. 0 Negate $\overline{\text{RSTOUT}}$ pin 1 Assert $\overline{\text{RSTOUT}}$ pin **Note:** External logic driving reset configuration data during reset needs to be considered when asserting the $\overline{\text{RSTOUT}}$ pin by setting FRCRSTOUT.
5–0	Reserved, must be cleared.

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	LOC	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, must be cleared.
5 SOFT	Software reset flag. Indicates the software caused the last reset. 0 Last reset not caused by software 1 Last reset caused by software
4 LOC	Loss-of-clock reset flag. Indicates PLL loss of reference clock caused the last reset. 0 Last reset not caused by PLL loss-of-clock 1 Last reset caused by PLL loss-of-clock
3 POR	Power-on reset flag. Indicates power-on reset caused the last reset. 0 Last reset not caused by power-on reset 1 Last reset caused by power-on reset
2 EXT	External reset flag. Indicates that the last reset was caused by an external device or circuitry asserting the external $\overline{\text{RESET}}$ pin. 0 Last reset not caused by external reset 1 Last reset caused by external reset
1 WDRCORE	Core watchdog timer reset flag. Indicates the core watchdog timer timeout caused the last reset. 0 Last reset not caused by watchdog timer timeout 1 Last reset caused by watchdog timer timeout
0 LOL	Loss-of-lock reset flag. Indicates the last reset state was caused by a PLL loss of lock. 0 Last reset not caused by loss of lock 1 Last reset caused by a loss of lock

10.4 Functional Description

10.4.1 Reset Sources

Table 10-5 defines the reset sources 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
Loss of clock	Asynchronous
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 immediately asserted 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. After approximately another 512 cycles, $\overline{\text{RSTOUT}}$ is negated and the device begins operation.

10.4.1.2 External Reset

Asserting $\overline{\text{RESET}}$ for at least four rising FB_CLK edges causes the device to recognize and latch the external reset request. After the $\overline{\text{RESET}}$ pin is negated and the PLL acquires lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 bus clock cycles. The device then exits reset and begins operation.

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

10.4.1.3 Core Watchdog Timer Reset

A core watchdog timer timeout causes the timer reset request to be recognized and latched. If $\overline{\text{RESET}}$ is negated and the PLL acquires lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 bus clock cycles. Then the device exits reset and begins operation.

10.4.1.4 Loss-of-Lock Reset

This reset condition occurs when the PLL loses lock. After the PLL acquires lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 bus clock cycles. The device then exits reset and resumes operation.

10.4.1.5 Loss-of-Clock Reset

This reset condition occurs when the PLL loses the reference clock. After the reference clock resumes and the PLL acquires lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 bus clock cycles. The device then exits reset and resumes operation.

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 bus clock 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 are numbered, and these numbers are referred to (within parentheses) in the flow description that follows. All cycle counts given are approximate.

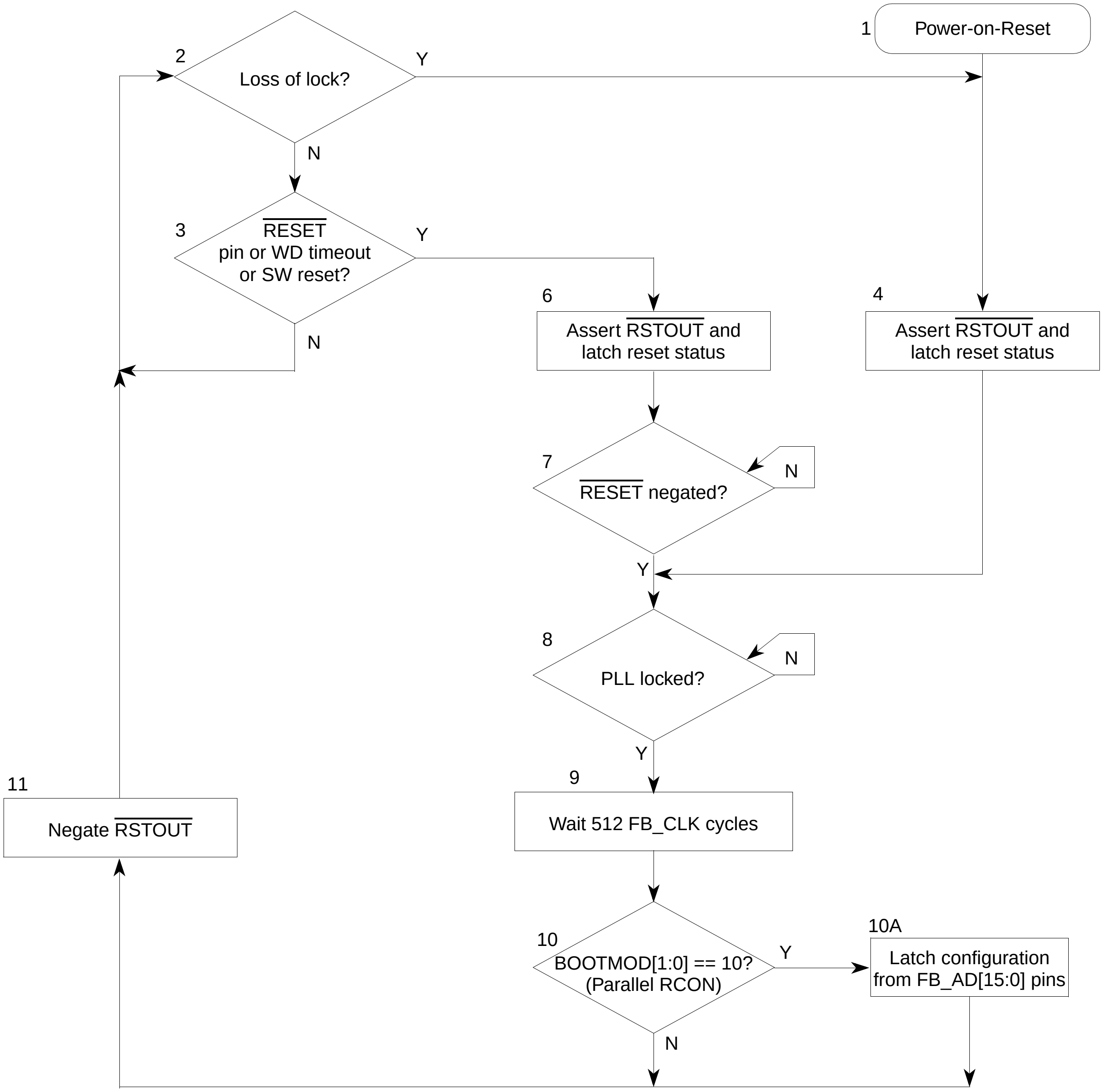


Figure 10-4. Reset Control Flow

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 device asserts the external $\overline{\text{RESET}}$ signal for at least four rising FB_CLK edges (3), if the watchdog timer times out, or if software requests a reset, the reset control logic latches the reset request internally. At this point the $\overline{\text{RSTOUT}}$ pin is asserted (6). (Even though the external $\overline{\text{RESET}}$ pin must be asserted for only four FB_CLK edges, it may take up to six clocks beyond $\overline{\text{RESET}}$ assertion for $\overline{\text{RSTOUT}}$ to assert.) The reset control logic waits until the $\overline{\text{RESET}}$ signal is negated (7) and for the PLL to attain lock (8) before waiting 512 FB_CLK cycles (9). The reset control logic may latch the chip configuration options from the FB_AD[15:0] pins (10, 10A). $\overline{\text{RSTOUT}}$ is then negated (11).

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

10.4.2.2 Asynchronous Reset Request

If reset is asserted by an asynchronous internal reset source, such as loss of lock (2) or power-on reset (1), the reset control logic asserts $\overline{\text{RSTOUT}}$ (4). The reset control logic waits for the PLL to attain lock (8) before waiting either 512 bus clock cycles (9). The reset control logic may then latch the chip configuration options from the FB_AD[15:0] pins (10, 10A). $\overline{\text{RSTOUT}}$ is then negated (11).

If a loss of lock occurs during the 512 bus clock count (9), the reset flow switches to (8) and waits for the PLL to lock before continuing.

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 (7).

If a loss-of-lock condition is detected during the 512 cycle wait, the reset sequence continues after a PLL lock (8).

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 reset sources are asserted after the RSR status bits have been latched (4 or 6), then the device is held in reset (9) until all sources have negated, and the subsequent sources will not be 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 masters and peripherals. It allows the privilege level of a master to be overridden, forcing it to user-mode privilege, and allows masters to be designated as trusted or untrusted. Peripherals may be programmed to require supervisor privilege level for access, may restrict access to a trusted master only, and may be write-protected.

The SCM's core watchdog timer (CWT) provides a means of preventing system lockup due to uncontrolled software loops via a special software service sequence. If periodic software servicing action does not occur, the CWT times out with a programmed response (system reset or interrupt) to allow recovery or corrective action to be taken.

Fault access reporting is also available within the SCM. The user can use these registers during the resulting interrupt service routine and perform an appropriate recovery.

11.1.2 Features

The SCM includes these distinctive features:

- Access control registers
  - Master privilege register (MPR)
  - Peripheral access control registers (PACRs)
- System control registers
  - Core watchdog control register (CWCR) for watchdog timer control
  - Core watchdog service register (CWSR) to service watchdog timer
  - SCM interrupt status register (SCMISR) to service a bus fault or watchdog interrupt
- 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 supports only 8-bit writes, etc. Attempted writes of a different size than the register width produce a bus error and no change to the targeted register.

Table 11-1. SCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_0000	Master Privilege Register (MPR)	32	R/W	0x7000_0007	11.2.1/11-3
0xFC00_0020	Peripheral Access Control Register A (PACRA)	32	R/W	0x5440_0400	11.2.2/11-4
0xFC00_0024	Peripheral Access Control Register B (PACRB)	32	R/W	0x0000_4400	11.2.2/11-4
0xFC00_0028	Peripheral Access Control Register C (PACRC)	32	R/W	0x4444_0444	11.2.2/11-4
0xFC00_002C	Peripheral Access Control Register D (PACRD)	32	R/W	0x4440_4444	11.2.2/11-4
0xFC00_0040	Peripheral Access Control Register E (PACRE)	32	R/W	0x4444_4444	11.2.2/11-4
0xFC00_0044	Peripheral Access Control Register F (PACRF)	32	R/W	0x4444_4444	11.2.2/11-4
0xFC00_0048	Peripheral Access Control Register G (PACRG)	32	R/W	0x4444_4444	11.2.2/11-4
Supervisor Access Only Registers					
0xFC04_0013	Wakeup Control Register (WCR)[1]	8	R/W	0x00	8.2.1/8-2
0xFC04_0016	Core Watchdog Control Register (CWCR)	16	R/W	0x0000	11.2.3/11-7
0xFC04_001B	Core Watchdog Service Register (CWSR)	8	R/W	Undefined	11.2.4/11-8
0xFC04_001F	SCM Interrupt Status Register (SCMISR)	8	R/W	0x00	11.2.5/11-9
0xFC04_0024	Burst Configuration Register (BCR)	32	R/W	0x0000_0000	11.2.6/11-10
0xFC04_0070	Core Fault Address Register (CFADR)	32	R	Undefined	11.2.7/11-10
0xFC04_0075	Core Fault Interrupt Enable Register (CFIER)	8	R/W	0x00	11.2.8/11-10
0xFC04_0076	Core Fault Location Register (CFLOC)	8	R	Undefined	11.2.9/11-11
0xFC04_0077	Core Fault Attributes Register (CFATR)	8	R	Undefined	11.2.10/11-11
0xFC04_007C	Core Fault Data Register (CFDTR)	32	R	Undefined	11.2.11/11-12

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

11.2.1 Master Privilege Register (MPR)

The MPR specifies five 4-bit fields defining the access-privilege level associated with a bus master in the device to the various peripherals listed in Table 11-4. The register provides one field per bus master.

Address: 0xFC00_0000 (MPR)

Access: User read/write

	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3	2	1	0
R	MPROT0	MPROT1	MPROT2	MPROT3	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 11-1. Master Privilege Register (MPR)

Each master is assigned depending on its connection to the various crossbar switch master ports.

Table 11-2. MPROT*n* Assignments

Crossbar Switch Port Number	MPROT*n*	Master
M0	MPROT0	ColdFire core
M1	MPROT1	eDMA controller
M2	MPROT2	FEC0
M3	MPROT3	FEC1
M4	MPROT4	eSDHC
M5	MPROT5	USB host
M6	MPROT6	USB On-the-Go
M7	MPROT7[1]	Reserved for factory test

[1] This field is located at MPR[3:0]. However, it is hardwired to 0111 and may not be altered.

The MPROT*n* field is defined as shown below.

	3	2	1	0
R	0	MTR	MTW	MPL
W				

Figure 11-2. MPROT*n* Fields

Table 11-3. MPROT*n* Field Descriptions

Field	Description
3	Reserved, must be cleared.
2 MTR	Master trusted for read. Determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.

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 Peripheral Access Control Registers (PACR*x*)

Each of the peripherals has a four-bit PACR*n* field which defines the access levels supported by the given module. Eight PACRs are grouped together to form a 32-bit PACR*x* register (PACRA-PACRG). At reset, the SCM (PACR0) does not allow access from untrusted masters, while the other peripherals do.

Address: 0xFC00_0020 (PACRA) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PACR0				PACR1				PACR2				0	0	0	0	0	0	0	0	PACR5				0	0	0	0	0	0	0	0
W																																
Reset	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 11-3. Peripheral Access Control Register A (PACRA)

Address: 0xFC00_0024 (PACRB) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PACR12				PACR13				0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 11-4. Peripheral Access Control Register B (PACRB)

Address: 0xFC00_0028 (PACRC) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PACR16				PACR17				PACR18				PACR19				0	0	0	0	PACR21				PACR22				PACR23			
W																																
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0

Figure 11-5. Peripheral Access Control Register C (PACRC)

Address: 0xFC00_002C (PACRD) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PACR24				PACR25				PACR26				0	0	0	0	PACR28				PACR29				PACR30				PACR31			
W																																
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0

Figure 11-6. Peripheral Access Control Register D (PACRD)

Address: 0xFC00_0040 (PACRE) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PACR32				PACR33				PACR34				PACR35				PACR36				PACR37				0	1	0	0	PACR39			
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 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 11-8. Peripheral Access Control Register F (PACRF)

Address: 0xFC00_0048 (PACRG) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PACR48				PACR49				PACR50				PACR51				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 11-9. Peripheral Access Control Register G (PACRG)

Each peripheral is assigned to its PACR*n* field:

Table 11-4. PACR*n* Assignments

Slot Number	PACR*n*	Peripheral
0	PACR0	SCM (MPR & PACRs)
1	PACR1	Crossbar switch
2	PACR2	FlexBus
5	PACR5	MPU
12	PACR12	FEC0
13	PACR13	FEC1
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	DSPI
24	PACR24	UART0
25	PACR25	UART1

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

Slot Number	PACR*n*	Peripheral
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	Edge port 0
37	PACR37	Edge port 1
39	PACR39	Codec
40	PACR40	CCM, reset controller, power management
41	PACR41	Pin multiplexing and control (GPIO)
42	PACR42	Real time clock (RTC)
43	PACR43	SIM (Smartcard interface)
44	PACR44	USB On-the-Go
45	PACR45	USB host
46	PACR46	SDRAM controller
47	PACR47	SSI
48	PACR48	PLL
49	PACR49	Random number generator (RNG)
50	PACR50	IIM
51	PACR51	eSDHC

The PACR*n* field is defined as:

	3	2	1	0
R	0	SP	WP	TP
W				

Figure 11-10. PACR*n* Fields

Table 11-5. PACR*n* Field Descriptions

Field	Description
3	Reserved, must be cleared.
2 SP	Supervisor protect. Determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The master privilege level must indicate supervisor access attribute, and the MPROT*n*[MPL] control bit for the master must be set. If not, access terminates with an error response and no peripheral access initiates.
1 WP	Write protect. Determines whether the peripheral allows write accesses 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, access terminates with an error response and no peripheral access initiates.
0 TP	Trusted protect. Determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access terminates with an error response and no peripheral access initiates.

11.2.3 Core Watchdog Control Register (CWCR)

The CWCR controls the software watchdog timer, time-out periods, and software watchdog timer interrupt. The register can be read or written at any time. At system reset, the software watchdog timer is disabled.

Address: 0xFC04_0016 (CWCR)

Access: Supervisor 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 11-11. Core Watchdog Control Register (CWCR)

Table 11-6. CWCR Field Descriptions

Field	Description
15 RO	Read-only control bit. 0 CWCR can be read or written. 1 CWCR is read-only. A system reset is required to clear this register. The setting of this bit is intended to prevent accidental writes of the CWCR from changing the defined core watchdog configuration.
14–9	Reserved, must be cleared.
8 CWRWH	Core watchdog run while halted. 0 Core watchdog timer stops counting if the core is halted. 1 Core watchdog timer continues to count even while the core is halted.
7 CWE	Core watchdog timer enable. 0 CWT is disabled. 1 CWT is enabled.

Table 11-6. CWCR Field Descriptions (continued)

Field	Description
6–5 CWRI	Core watchdog reset/interrupt. 00 If a time-out occurs, the CWT generates an interrupt to the core. Refer to Chapter 15, “Interrupt Controller Modules,” for details on setting its priority level. 01 The first time-out generates an interrupt to the processor, and if not serviced, a second time-out generates a system reset and sets the RSR[WDRCORE] flag in the reset controller. 10 If a time-out occurs, the CWT generates a system reset and RSR[WDRCORE] in the reset controller is set. 11 The CWT functions in a window mode of operation. For this mode, the servicing of the CWSR must occur during the last 25% of the time-out period. Any writes to the CWSR during the first 75% of the time-out period generate an immediate system reset. Likewise, if the CWSR is not serviced during the last 25% of the time-out period, a system reset is generated. For any type of reset response, the RSR[WDRCORE] flag is set.
4–0 CWT	Core watchdog time-out period. Selects the time-out period for the CWT. At reset, this field is cleared selecting the minimum time-out period, but the CWT is disabled because CWCR[CWE] is cleared at reset. If CWCR[CWT] is *n*, the time-out period equals 2^n system clock cycles. However, if *n* is less than 8, the time-out period is forced to 2^8. 0x00 2^8 ... 0x08 2^8 0x09 2^9 ... 0x1F 2^{31}

11.2.4 Core Watchdog Service Register (CWSR)

The software watchdog service sequence must be performed using the CWSR as a data register to prevent a CWT time-out. The service sequence requires two writes to this data register: a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the CWT time-out, but any number of instructions can be executed between the two writes. If the CWT has already timed out, writing to this register has no effect in negating the CWT interrupt or reset. Figure 11-12 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: Supervisor read/write

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

Figure 11-12. Core Watchdog Service Register (CWSR)

11.2.5 SCM Interrupt Status Register (SCMISR)

For certain values in the CWCR[CWRI] field, the CWT generates an interrupt response to a time-out. For these configurations, the SCMISR provides a program visible interrupt request from the watchdog timer. During the interrupt service routine which handles this interrupt, the source must be explicitly cleared by writing a 0x01 to the SCMISR.

The SCMISR also indicates system bus fault errors. An interrupt is sent only to the interrupt controller when the CFIER[ECFEI] bit is set. The SCMISR[CFEI] bit flags fault errors independent of the CFIER[ECFEI] setting. Therefore, if CFEI is set prior to setting ECFEI, an interrupt is requested immediately after ECFEI is set.

Address: 0xFC04_001F (SCMISR) Access: Supervisor 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 11-13. SCM Interrupt Status Register (SCMISR)

Table 11-7. SCMISR Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1 CFEI	Core fault error interrupt flag. Indicates if a bus fault has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect. 0 No bus error. 1 A bus error has occurred. The faulting address, attributes (and possibly write data) are captured in the CFADR, CFATR, and CFDTR registers. The error interrupt is enabled only if CFLOC[ECFEI] is set. **Note:** This bit reports core faults regardless of the setting of CFIER[ECFEI]. Therefore, if the error interrupt is disabled and a core fault occurs, this bit is set. Then, if the error interrupt is subsequently enabled, an interrupt is immediately requested. To prevent an undesired interrupt, clear the captured error by writing one to CFEI before enabling the interrupt.
0 CWIC	Core watchdog interrupt flag. Indicates whether an CWT interrupt has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect. 0 No CWT interrupt has occurred. 1 CWT interrupt has occurred.

11.2.6 Burst Configuration Register (BCR)

The BCR register enables or disables the USB host and USB On-the-Go modules for bursting to/from the crossbar switch slave modules. There is an enable field for the slaves, and either direction (read and write) is supported via the GBR and GBW bits.

11.2.7 Core Fault Address Register (CFADR)

The CFADR is a read-only register indicating the address of the last core access terminated with an error response.

Address: 0xFC04_0070 (CFADR) Access: Supervisor read-only

<table>
<tr><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td colspan="32">ADDR</td></tr>
<tr><td>W</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Reset</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td><td>–</td></tr>
</table>

Figure 11-14. Core Fault Address Register (CFADR)

Table 11-8. CFADR Field Descriptions

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

11.2.8 Core Fault Interrupt Enable Register (CFIER)

The CFIER register enables the system bus-error interrupt. See Chapter 15, "Interrupt Controller Modules," for more information of the interrupt controller.

Address: 0xFC04_0075 (CFIER) Access: Supervisor read/write

<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td rowspan="2">ECFEI</td></tr>
<tr><td>W</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Reset:</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 11-15. Core Fault Interrupt Enable Register (CFIER)

Table 11-9. CFIER Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 ECFEI	Enable core fault error interrupt. 0 Do not generate an error interrupt on a faulted system bus cycle. 1 Generate an error interrupt to the interrupt controller on a faulted system bus cycle.

11.2.9 Core Fault Location Register (CFLOC)

The read-only CFLOC register indicates the exact location within the device of the captured fault information.

Address: 0xFC04_0076 (CFLOC) Access: Supervisor 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 11-16. Core Fault Location Register (CFLOC)

Table 11-10. CFLOC Field Descriptions

Field	Description
7 LOC	The location of the last captured fault. 0 Error occurred on the internal bus. 1 Error occurred within the core.
6–0	Reserved, must be cleared.

11.2.10 Core Fault Attributes Register (CFATR)

The read-only CFATR register captures the processor's attributes of the last faulted core access to the system bus.

Address: 0xFC04_0077 (CFATR) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R	WRITE	SIZE			CACHE	0	MODE	TYPE
W								
Reset:	–	–	–	–	–	–	–	–

Figure 11-17. Core Fault Attributes Register (CFATR)

Table 11-11. 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.

Table 11-11. CFATR Field Descriptions (continued)

Field	Description
3 CACHE	Indicates if last faulted core access was cacheable. 0 Non-cacheable 1 Cacheable
2	Reserved, must be cleared.
1 MODE	Indicates the mode the device was in during the last faulted core access. 0 User mode 1 Supervisor mode
0 TYPE	Defines the type of last faulted core access. 0 Instruction 1 Data

11.2.11 Core Fault Data Register (CFDTR)

The CFDTR is a read-only register for capturing the data associated with the last faulted processor write data access from the device’s internal bus. The CFDTR is valid only for faulted internal bus-write accesses, CFLOC[LOC] is cleared.

Address: 0xFC04_007C (CFDTR) Access: Supervisor 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 11-18. Core Fault Data Register (CFDTR)

Table 11-12. CFDTR Field Descriptions

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

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:

- An interrupt may be generated to the core.
- An immediate system reset.
- Upon the first time-out, a watchdog timer interrupt is asserted. If this time-out condition is not serviced before a second time-out occurs, the CWT asserts a system reset. This configuration supports a more graceful response to watchdog time-outs.
- In addition to these three basic modes of operation, the CWT also supports a windowed mode of operation. In this mode, the time-out period is divided into four equal segments and the entire service sequence of the CWT must occur during the last segment (last 25% of the time-out period). If the timer is serviced anytime (any write to the CWSR register) in the first 75% of the time-out period, an immediate system reset occurs.

To prevent the core watchdog timer from interrupting or resetting, the CWSR register must be serviced by performing the following sequence:

1. Write 0x55 to CWSR.
2. Write 0xAA to CWSR.

Both writes must occur in order before the time-out, but any number of instructions can execute between the two writes. This allows interrupts and exceptions to occur, if necessary, between the two writes.

NOTE

If the CWT is enabled and has not timed out, any write of a data value other than 0x55 or 0xAA causes an immediate system reset, regardless of the value in the CWCR[CWRI] field.

The timer value is constantly compared with the time-out period specified by CWCR[CWT], and any write to the CWCR register resets the watchdog timer. In addition, a write-once control bit in the CWCR sets the CWCR to read-only to prevent accidental updates to this control register from changing the desired system configuration. After this bit, CWCR[RO], is set, a system reset is required to clear it.

For certain values in the CWCR[CWRI] field, the CWT generates an interrupt response to a time-out. For these configurations, the SCMISR register provides a program visible interrupt request from the watchdog timer. During the interrupt service routine which handles this interrupt, the source must be explicitly cleared by writing a 0x01 to the SCMISR.

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 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 MCF5301*x* devices have up to eight masters and three slaves (8Mx3S) connected to the crossbar switch. The eight masters are the ColdFire core, eDMA controller, FECs, USB host, USB OTG modules, eSDHC, and a reserved master for factory test. The slaves are FlexBus/SDRAM controller, SRAM controller backdoor,and the peripheral bus controller.

Figure 12-1 is a block diagram of the MCF5301*x* family bus architecture showing the crossbar switch configuration.

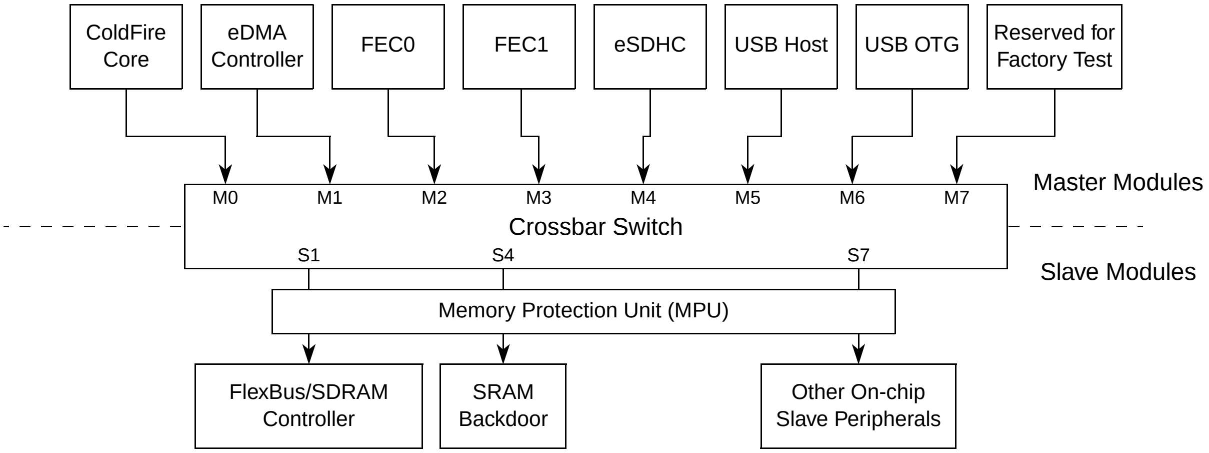


Figure 12-1. Bus Architecture Block Diagram

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 0	
Master 3 (M3)	Fast Ethernet controller 1	
Master 4 (M4)	eSDHC	
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 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 11.2.5, “SCM Interrupt Status Register (SCMISR).”

The slave registers also feature a bit that, when set, prevents the registers from being written. The registers remain readable, but future write attempts have no effect on the registers and are terminated with a bus error response to the master initiating the write. The core, for example, takes a bus error interrupt.

Table 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	0x7654_3210	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	0x7654_3210	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_4700	Priority Register Slave 7 (XBS_PRS7)	32	R/W	0x7654_3210	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 11.2.5, "SCM Interrupt Status Register (SCMISR)") and the XBS_PRS*n* is not updated.

Address: 0xFC00_4100 (XBS_PRS1)
0xFC00_4400 (XBS_PRS4)
0xFC00_4700 (XBS_PRS7)

Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	M7			0	M6			0	M5			0	M4			0	M3			0	M2			0	M1			0	M0		
W																																
Reset	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0	0	0	1	1	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 priority when accessing the slave port. 111 This master has level 8 (lowest) priority when accessing the slave port.
27	Reserved, must be cleared.
26–24 M6	Master 6 (USB OTG) priority. See M7 description.
23	Reserved, must be cleared.
22–20 M5	Master 5 (USB Host) priority. See M7 description.

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

Field	Description
19	Reserved, must be cleared.
18–16 M4	Master 4 (eSDHC) priority. See M7 description.
15	Reserved, must be cleared.
14–12 M3	Master 3 (FEC1) priority. See M7 description.
11	Reserved, must be cleared.
10–8 M2	Master 2 (FEC0) priority. See M7 description.
7	Reserved, must be cleared.
6–4 M1	Master 1 (eDMA) priority. See M7 description.
3	Reserved, must be cleared.
2–0 M0	Master 0 (ColdFire core) priority. See M7 description.

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)
0xFC00_4410 (XBS_PRS4)
0xFC00_4710 (XBS_PRS7)

Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RO[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ARB	0	0	PCTL		0	PARK		
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] After this bit is set, only a hardware reset clears it.

Figure 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, only a hardware reset clears it. 0 Both of the slave port's registers are writeable. 1 Both of the slave port's registers are read-only and cannot be written (attempted writes have no effect on the registers and result in a bus error response).
30–9	Reserved, must be cleared.

Table 12-4. XBS_CRS*n* Field Descriptions (continued)

Field	Description
8 ARB	Arbitration Mode. Selects the arbitration policy for the slave port. 0 Fixed priority 1 Round robin (rotating) priority
7–6	Reserved, must be cleared.
5–4 PCTL	Parking control. Determines the slave port's parking control. The low-power park feature results in an overall power savings if the slave port is not saturated; however, this forces an extra latency clock when any master tries to access the slave port while not in use because it is not parked on any master. 00 When no master makes a request, the arbiter parks the slave port on the master port defined by the PARK bit field. 01 When no master makes a request, the arbiter parks the slave port on the last master to be in control of the slave port. 10 When no master makes a request, the slave port is not parked on a master and the arbiter drives all outputs to a constant safe state. 11 Reserved.
3	Reserved, must be cleared.
2–0 PARK	Park. Determines which master port the current slave port parks on when no masters are actively making requests and the PCTL bits are cleared. 000 Park on master port M0 (ColdFire Core) 001 Park on master port M1 (eDMA Controller) 010 Park on master port M2 (FEC0) 011 Park on master port M3 (FEC1) 100 Park on master port M4 (eSDHC) 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 11.2.5, "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
Memory Protection Unit (MPU)

13.1 Introduction

The memory protection unit (MPU) provides hardware access control for all memory references generated in the device.

13.2 Overview

The MPU concurrently monitors all system bus transactions and evaluates their appropriateness using pre-programmed region descriptors that define memory spaces and their access rights. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

13.2.1 Block Diagram

The location of the MPU within the processor is shown in Figure 13-1.

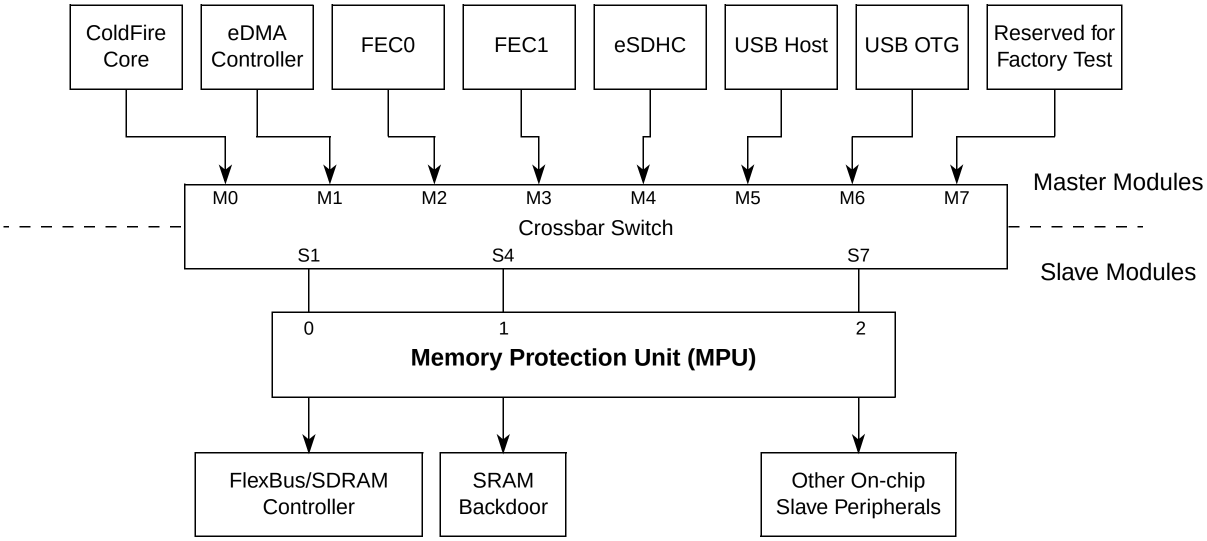


Figure 13-1. MPU Connection in Microprocessor

A simplified block diagram of the MPU module is shown in Figure 13-2. The hardware's two-dimensional connection matrix is clearly visible with the basic access evaluation macro shown as the replicated submodule block. The crossbar switch slave ports are shown on the left, the region descriptor registers in the middle, and the peripheral bus interface on the right side. The evaluation macro contains two magnitude comparators connected to the start and end address registers from each region descriptor as well as the combinational logic blocks to determine the region hit and the access protection error. For details of the access evaluation macro, see Section 13.5.1, “Access Evaluation Macro.”

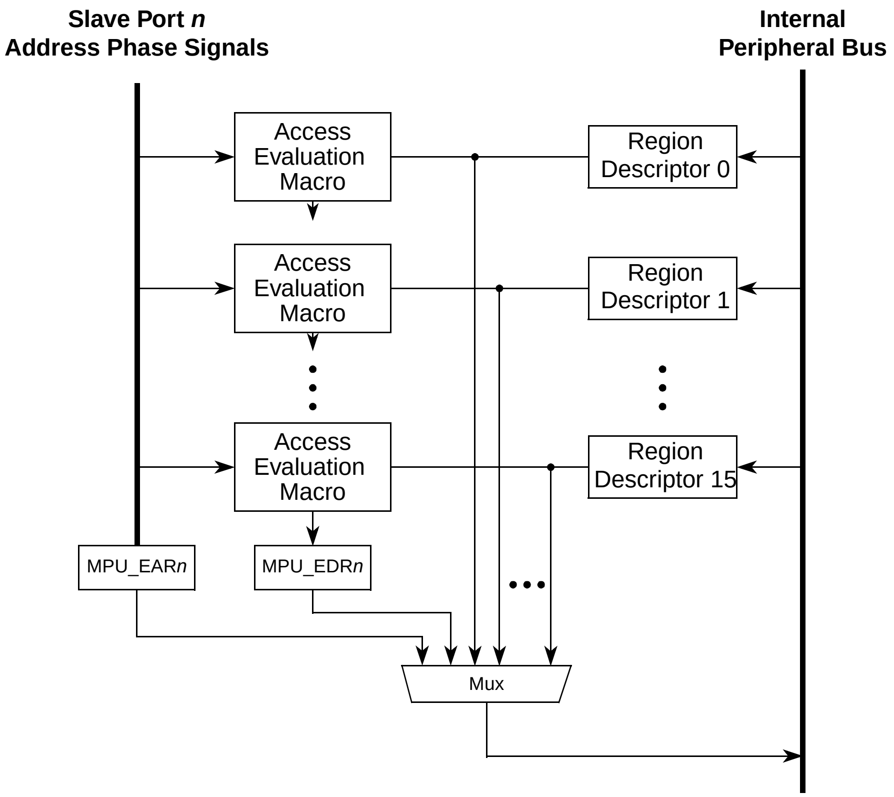


Figure 13-2. MPU Block Diagram

13.2.2 Features

The memory protection unit implements a two-dimensional hardware array of memory region descriptors and the crossbar slave ports to continuously monitor the legality of every memory reference generated by each bus master in the system. The feature set includes:

- 16 program-visible 128-bit region descriptors, accessible by four longwords each
  - — Each region descriptor defines a modulo-32 byte space, aligned anywhere in memory
    - – Region sizes can vary from 32 bytes to 4 Gbytes
  - — Two access control permissions defined in a single descriptor word
    - – Masters 0–3: read, write, and execute attributes for supervisor and user accesses
    - – Masters 4–6: read and write attributes
  - — Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
  - — Alternate programming model view of the access control permissions word
  - — Priority given to granting permission over denying access for overlapping region descriptors
- Detects access protection errors if a memory reference does not hit in any memory region, or if the reference is illegal in hit memory regions. If an access error occurs, the reference is terminated with an error response, and the MPU inhibits the bus cycle being sent to the targeted slave device.

- 64-bit error registers (one per slave port) capture the last faulting address, attributes, and other information
- Global MPU enable/disable control bit

13.2.3 Modes of Operation

The MPU module does not support any special modes of operation.

13.3 External Signal Description

The MPU module does not include any external interface.

13.4 Memory Map/Register Definition

The programming model is partitioned into three groups: control/status registers, the data structure containing the region descriptors, and the alternate view of the region descriptor access control values.

The programming model can only be referenced using 32-bit (longword) accesses. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (a write to a read-only register, or a read of a write-only register) generate an error termination.

Table 13-1. MPU Memory Map

Address	Register Name	Register Description	Width (bits)	Access	Reset Value	Section/ Page
0xFC01_4000	MPU_CESR	MPU control/error status register	32	R/W	0x------00	13.4.1/13-4
0xFC01_4010	MPU_EAR0	MPU error address register, slave 0	32	R	*	13.4.2/13-5
0xFC01_4014	MPU_EDR0	MPU error detail register, slave 0	32	R	*	13.4.3/13-5
0xFC01_4018	MPU_EAR1	MPU error address register, slave 1	32	R	*	13.4.2/13-5
0xFC01_401c	MPU_EDR1	MPU error detail register, slave 1	32	R	*	13.4.3/13-5
0xFC01_4020	MPU_EAR2	MPU error address register, slave 2	32	R	*	13.4.2/13-5
0xFC01_4024	MPU_EDR2	MPU error detail register, slave 2	32	R	*	13.4.3/13-5
0xFC01_4400 + *n* × 0x10	MPU_RGD*n*	MPU region descriptor *n* *n* = 0–15	128	R/W	*	13.4.4/13-6
0xFC01_4800 + *n* × 0x4	MPU_RGDAAC*n*	MPU RGD alternate access control *n* *n* = 0–15	32	R/W	*	13.4.4.5/13-10

13.4.1 MPU Control/Error Status Register (MPU_CESR)

MPU_CESR provides one byte of error status plus three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Address: 0xFC01_4000 Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SPERR			0	0	0	0	0	1	0	0	0	HRL				NSP				NRGD				0	0	0	0	0	0	0	VLD
W	w1c	w1c	w1c																													
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0

Figure 13-3. MPU Control/Error Status Register (MPU_CESR)

Table 13-2. MPU_CESR Field Descriptions

Field	Description
31–29 SPERR	Slave port *n* error. Indicates a captured error in MPU_EAR*n* and MPU_EDR*n*. This bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared by writing one to it. If another error is captured at the exact same cycle as the write, the flag remains set. A find-first-one instruction (or equivalent) can detect the presence of a captured error. The correlation between the MPU slave ports and crossbar switch slaves is: SPERRn | MPU Port # | Crossbar Port # | Slave 2 | 0 | 1 | Flexbus/SDRAM 1 | 1 | 4 | SRAM backdoor 0 | 2 | 7 | Other slave peripherals 0 MPU_EAR*n* and MPU_EDR*n* do not contain a captured error 1 MPU_EAR*n* and MPU_EDR*n* contain a captured error
28–24	Reserved, must be cleared.
23	Reserved, must be set.
22–20	Reserved, must be cleared.
19–16 HRL	Hardware revision level. Specifies the MPU's hardware and definition revision level. It can be read by software to determine the functional definition of the module.
15–12 NSP	Number of slave ports. Specifies the number of slave ports connected to the MPU. For this device, this field is set to 0x3.
11–8 NRGD	Number of region descriptors. Specifies the number of region descriptors implemented in the MPU. For this device, this field is set to 0x2 to indicate 16 region descriptors are supported.
7–1	Reserved, must be cleared.
0 VLD	Valid. Global enable/disable for the MPU. 0 MPU is disabled. All accesses from all bus masters are allowed. 1 MPU is enabled.

13.4.2 MPU Error Address Register, Slave Port *n* (MPU_EAR*n*)

When the MPU detects an access error on slave port *n*, the 32-bit reference address is captured in this read-only register and the corresponding bit in MPU_CESR[SPERR] set. Additional information about the faulting access is captured in the corresponding MPU_EDR*n* at the same time. This register and the corresponding MPU_EDR*n* contain the most recent access error; there are no hardware interlocks with MPU_CESR[SPERR], as the error registers are always loaded upon the occurrence of each protection violation.

Address: 0xFC01_4010 (MPU_EAR0)
0xFC01_4018 (MPU_EAR1)
0xFC01_4020 (MPU_EAR2)

Access: 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	EADDR																															
W																																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 13-4. MPU Error Address Register, Slave Port *n* (MPU_EAR*n*)

Table 13-3. MPU_EARn Field Descriptions

Field	Description
31–0 EADDR	Error Address. Indicates the reference address from slave port *n* that generated the access error.

13.4.3 MPU Error Detail Register, Slave Port *n* (MPU_EDR*n*)

When the MPU detects an access error on slave port *n*, 32 bits of error detail are captured in this read-only register and the corresponding bit in MPU_CESR[SPERR] is set. Information on the faulting address is captured in the corresponding MPU_EAR*n* register at the same time. This register and the corresponding MPU_EAR*n* register contain the most recent access error; there are no hardware interlocks with MPU_CESR[SPERR] as the error registers are always loaded upon the occurrence of each protection violation.

Address: 0xFC01_4014 (MPU_EDR0)
0xFC01_401C (MPU_EDR1)
0xFC01_4024 (MPU_EDR2)

Access: 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	EACD																0	0	0	0	0	0	0	0	EMN				EATTR			ERW
W																																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-

Figure 13-5. MPU Error Detail Register, Slave Port *n* (MPU_EDR*n*)

Table 13-4. MPU_EDR*n* Field Descriptions

Field	Description
31–16 EACD	Error access control detail. Indicates the region descriptor with the access error. If MPU_EDR*n* contains a captured error and EACD is cleared, an access did not hit in any region descriptorIf only a single EACD bit is set, the protection error was caused by a single non-overlapping region descriptor. If two or more EACD bits are set, the protection error was caused by an overlapping set of region descriptors.
7–4 EMN	Error master number. Indicates the bus master that generated the access error.
3–1 EATTR	Error attributes. Indicates attribute information about the faulting reference. 000 User mode, instruction access 001 User mode, data access 010 Supervisor mode, instruction access 011 Supervisor mode, data access **Note:** All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (011).
0 ERW	Error read/write. Indicates the access type of the faulting reference. 0 Read 1 Write

13.4.4 MPU Region Descriptor *n* (MPU_RGD*n*)

Each 128-bit (16 byte) region descriptor specifies a given memory space and the access attributes associated with that space.The region descriptors are organized sequentially in the MPU's memory map as four 32-bit words.

Since the region descriptor is a 128-bit entity, there are potential coherency issues since multiple writes are required to update the entire descriptor. It is recommended to follow the following write sequence:

1. MPU_RGD*n*.Word0
2. MPU_RGD*n*.Word1
3. MPU_RGD*n*.Word2
4. MPU_RGD*n*.Word3

The MPU automatically clears MPU_RGD*n*.Word3[VLD] on any writes to descriptor words 0, 1, or 2. Writes to MPU_RGD*n*.Word3 set/clear the valid bit in a normal manner.

Since it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* as stores to these locations do not affect the descriptor's valid bit.

13.4.4.1 MPU Region Descriptor *n*, Word 0 (MPU_RGDn.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this register clear the region descriptor's valid bit (MPU_RGD*n*.Word3[VLD]).

Address: 0xFC01_4400 + (16×*n*) (MPU_RGD*n*.Word0) Access: R/W

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

Figure 13-6. MPU Region Descriptor, Word 0 Register (MPU_RGDn.Word0)

Table 13-5. MPU_RGD*n*.Word0 Field Descriptions

Field	Description
31–5 SRTADDR	Start address. Defines the most significant bits of the 0-modulo-32 byte start address of the memory region.
4–0	Reserved, must be cleared.

13.4.4.2 MPU Region Descriptor *n*, Word 1 (MPU_RGD*n*.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this register clear the region descriptor's valid bit (MPU_RGD*n*.Word3[VLD]).

Address: 0xFC01_4404 + (16×*n*) (MPU_RGD*n*.Word1) Access: R/W

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

Figure 13-7. MPU Region Descriptor, Word 1 Register (MPU_RGDn.Word1)

Table 13-6. MPU_RGDn.Word1 Field Descriptions

Field	Description
31–5 ENDADDR	End address. Defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR ≥ SRTADDR; it is software's responsibility to properly load these region descriptor fields.
4–0	Reserved, must be set.

13.4.4.3 MPU Region Descriptor *n*, Word 2 (MPU_RGD*n*.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges depend on two broad classifications of bus masters:

- Bus masters 0–3 have a 5-bit field defining separate privilege rights for user and supervisor mode accesses.
- Bus masters 4–6 are limited to separate read and write permissions.

For the privilege rights of bus masters 0–3, there are three flags associated with this function:

- Read (r) refers to accessing the referenced memory address using an operand (data) fetch
- Write (w) refers to updating the referenced memory address using a store (data) instruction
- Execute (x) refers to reading the referenced memory address using an instruction fetch

Writes to MPU_RGD*n*.Word2 clear the region descriptor's valid bit (MPU_RGD*n*.Word3[VLD]). If only updating the access controls, write to MPU_RGDAAC*n* instead. Stores to these locations do not affect the descriptor's valid bit.

Address: 0xFC01_4400 + (16*n) + 0x8 (MPU_RGD*n*.Word2) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	0	M3SM		M3UM			0	M2SM
W												r	w	x		
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	M2SM	M2UM			0	M1SM		M1UM			0	M0SM		M0UM		
W		r	w	x				r	w	x				r	w	x
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 13-8. MPU Region Descriptor, Word 2 Register (MPU_RGD*n*.Word2)

Table 13-7. MPU_RGD*n*.Word2 Field Descriptions

Field	Description
31–30	Reserved, must be cleared.
29 M6RE	Bus master 6 (USB OTG) read enable. **Note:** 0Bus master 6 reads terminate with an access error and the read is not performed **Note:** 1Bus master 6 reads allowed
28 M6WE	Bus master 6 (USB OTG) write enable. 0 Bus master 6 writes terminate with an access error and the write is not performed 1 Bus master 6 writes allowed
27 M5RE	Bus master 5 (USB host) read enable. 0 Bus master 5 reads terminate with an access error and the read is not performed 1 Bus master 5 reads allowed
26 M5WE	Bus master 5 (USB host) write enable. 0 Bus master 5 writes terminate with an access error and the write is not performed 1 Bus master 5 writes allowed
25 M4RE	Bus master 4 (eSDHC) read enable. 0 Bus master 4 reads terminate with an access error and the read is not performed 1 Bus master 4 reads allowed
24 M4WE	Bus master 4 (eSDHC) write enable. 0 Bus master 4 writes terminate with an access error and the write is not performed 1 Bus master 4 writes allowed
23	Reserved, must be cleared.

Table 13-7. MPU_RGD*n*.Word2 Field Descriptions (continued)

Field	Description
22–21 M3SM	Bus master 3 (FEC1) supervisor mode access control. Defines the access controls for bus master 3 in supervisor mode. 00 r/w/x; read, write and execute allowed 01 r/x; read and execute allowed, but no write 10 r/w; read and write allowed, but no execute 11 Same as user mode defined in M3UM
20–18 M3UM	Bus master 3 (FEC1) user mode access control. Defines the access controls for bus master 3 in user mode. M3UM consists of three independent bits, enabling read (r), write (w), and execute (x)permissions. 1 Allows the given access type to occur 0 An attempted access of that mode may be terminated with an access error (if not allowed by another descriptor) and the access not performed.
17	Reserved, must be cleared.
16–15 M2SM	Bus master 2 (FEC0) supervisor mode access control. See M3SM description.
14–12 M2UM	Bus master 2 (FEC0) user mode access control. See M3UM description.
11	Reserved, must be cleared.
10–9 M1SM	Bus master 1 (eDMA) supervisor mode access control. See M3SM description.
8–6 M1UM	Bus master 1 (eDMA) user mode access control. See M3UM description.
5	Reserved, must be cleared.
4–3 M0SM	Bus master 0 (Core) supervisor mode access control. See M3SM description.
2–0 M0UM	Bus master 0 (Core) user mode access control. See M3UM description.

13.4.4.4 MPU Region Descriptor *n*, Word 3 (MPU_RGD*n*.Word3)

The fourth word of the MPU region descriptor contains the region descriptor’s valid bit.

Address: 0xFC01_440C + (16 × n) (MPU_RGD*n*.Word3) Access: Read/write

	31	30	29	28	27	26	25	24	23	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	VLD
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 13-9. MPU Region Descriptor, Word 3 Register (MPU_RGD*n*.Word3)

Table 13-8. MPU_RGD*n*.Word3 Field Descriptions

Field	Description
31–1	Reserved, must be cleared.
0 VLD	Valid. Signals the region descriptor is valid. Any write to MPU_RGD*n*.Word0–2 clears this bit. 0 Region descriptor is invalid 1 Region descriptor is valid

13.4.4.5 MPU Region Descriptor Alternate Access Control *n* (MPU_RGDAAC*n*)

Since software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is available. Writing to this register does not affect the descriptor's valid bit.

Address: 0xFC01_4800 + (4 × *n*) (MPU_RGDAAC*n*) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	0	M3SM		M3UM			0	M2SM
W												r	w	x		
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	M2SM	M2UM			0	M1SM		M1UM			0	M0SM		M0UM		
W		r	w	x				r	w	x				r	w	x
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 13-10. MPU RGD Alternate Access Control n (MPU_RGDAAC*n*)

MPU_RGDAAC*n* field descriptions are the same as MPU_RGD*n*.Word2. See Table 13-7.

13.5 Functional Description

In this section, the functional operation of the MPU is detailed, including the operation of the access evaluation macro and the handling of error-terminated bus cycles.

13.5.1 Access Evaluation Macro

The basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. As shown in Figure 13-11, the access evaluation macro inputs the crossbar bus address phase signals and the contents of a region descriptor (RGD*n*) and performs two major functions: region hit determination and detection of an access protection violation. Figure 13-11 is not intended to be a schematic of the actual access evaluation macro, but rather a generalized block diagram showing the major functions included in this logic block.

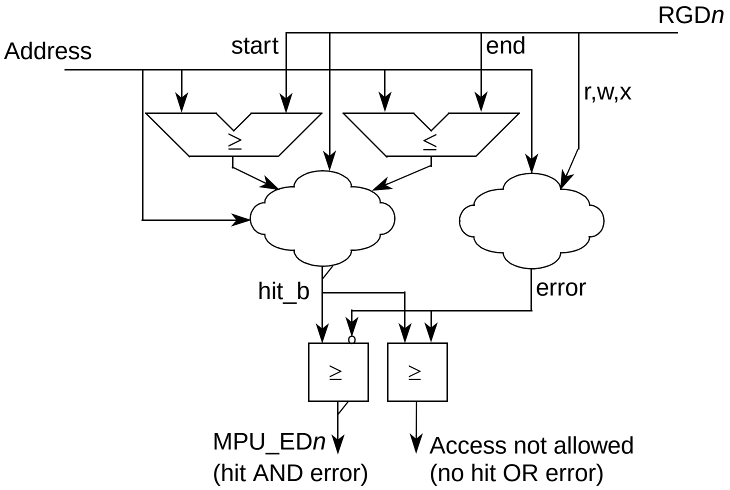


Figure 13-11. MPU Access Evaluation Macro

13.5.1.1 Hit Determination

To determine if the current reference hits in the given region, two magnitude comparators are used with the region’s start and end addresses. The boolean equation for this portion of the hit determination is:

```
region_hit = ((addr[31:5] ≥ MPU_RGDn.Word0[SRTADDR]) & (aaddr[31:5] ≤ MPU_RGDn.Word0[SRTADDR]))
                 & MPU_RGDn.Word3[VLD]
```

where addr is the current reference address, MPU_RGD*n*.Word0[SRTADDR] and MPU_RGD*n*.Word0[ENDADDR] are the start and end addresses, and MPU_RGD*n*.Word3[VLD] is the valid bit. There are no hardware checks to verify that ENDADDR ≥ SRTADDR, and it is your responsibility to load appropriate values into these fields of the region descriptor.

As shown in Figure 13-11, the access evaluation macro actually forms the logical complement (hit_b) of the combined region_hit and pid_hit boolean equations.

13.5.1.2 Privilege Violation Determination

While the access evaluation macro is determining region hit, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the master and supervisor/user mode signals, a set of effective permissions is generated from the appropriate fields in the region descriptor. The protection violation logic then evaluates the access against the effective permissions using the specification shown below.

Table 13-9. Protection Violation Definition

Description	MxUM			Protection Violation?
	r	w	x	
Instruction fetch read	—	—	0	Yes, no execute permission
	—	—	1	No, access is allowed

Table 13-9. Protection Violation Definition (continued)

Description	MxUM			Protection Violation?
	r	w	x	
Data read	0	—	—	Yes, no read permission
	1	—	—	No, access is allowed
Data write	—	0	—	Yes, no write permission
	—	1	—	No, access is allowed

13.5.2 Putting It All Together and Error Terminations

For each slave port monitored, the MPU performs a reduction-AND of all the individual terms from each access evaluation macro. This expression then terminates the bus cycle with an error and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, a protection error is reported.

As shown in the third condition, granting permission is a higher priority than denying access for overlapping regions. This approach is more flexible to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see Section 13.7, “Application Information”.

13.6 Initialization Information

The reset state of MPU_CESR[VLD] disables the entire module, and accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] is cleared.

At system startup, load the appropriate number of region descriptors (including setting MPU_RGD*n*.Word3[VLD]) before setting MPU_CESR[VLD], which enables the module. This approach enables all the loaded region descriptors simultaneously.

NOTE

If a memory reference does not hit in any region descriptor, the attempted access is terminated with an error. A region descriptor must be set to allow access to the MPU programming model if further changes are needed.

13.7 Application Information

In an operational system, interfacing with the MPU is generally classified into the following activities:

- Creating a new memory region—Load the appropriate region descriptor into an available MPU_RGD*n*, using four sequential 32-bit (longword) writes. As discussed in Section 13.4.4.4, “MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)”, the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues with the multi-cycle descriptor writes. (Clearing MPU_RGD*n*.Word3[VLD] deletes/removes an existing memory region.)
- Altering only access privileges—To not affect the valid bit, write to the alternate version of the access control word (MPU_RGDAAC*n*), so there are no coherency issues involved with the update. When the write completes, the memory region’s access rights switch instantaneously to the new value.
- Changing a region’s start and end addresses—Write a minimum of three words to the region descriptor (MPU_RGD*n*.Word{0,1,3}). Word 0 and 1 redefine the start and end addresses, respectively. Word 3 re-enables the region descriptor valid bit. In most situations, all four words of the region descriptor are rewritten.
- Accessing the MPU—Allocate a region descriptor to restrict MPU access to supervisor mode from a specific master.

  Detecting an access error—The current bus cycle is terminated with an error response and MPU_EAR*n* and MPU_EDR*n* capture information on the faulting reference. The masters on this device respond with a bus error interrupt in the SCM or the master’s specific error interrupt. See Section 11.2.5, “SCM Interrupt Status Register (SCMISR),” for details of the bus error interrupt. The processor can retrieve the captured error address and detail information simply be reading MPU_E{A,D}R*n*. MPU_CESR[SPERR] signals which error registers contain captured fault data.
- Overlapping region descriptors—Applying overlapping regions often reduces the number of descriptors required for a given set of access controls. In the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator).

  The following dual-core system example contains four bus masters: the two processors (CP0, CP1) and two DMA engines (DMA1, a traditional data movement engine transferring data between RAM and peripherals and DMA2, a second engine transferring data to/from the RAM only). Consider the following region descriptor assignments:

Region Description	RGD*n*	CP0	CP1	DMA1	DMA2	
CP0 code	0	rwx	r--	—	—	Flash
CP1 code	1	r--	rwx	—	—	
CP0 data & stack	2	rw-	—	—	—	RAM
CP0 → CP1 shared data	3	r--	r--	—	—	
CP1 → CP0 shared data						
CP1 data & stack	4	—	rw-	—	—	
Shared DMA data	5	rw-	rw-	rw	rw	
MPU	6	rw-	rw-	—	rw	Peripheral space
Peripherals	7	rw-	rw-	rw	—	

Figure 13-12. Overlapping Region Descriptor Example

In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and IPS peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

Of particular interest are the two overlapping spaces: region descriptors 2 & 3 and 3 & 4.

The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (rw- | r--) = (rw-) permissions, while CP1 has (--- | r--) = (r--) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (r-- | ---) = (r--) permission, while CP1 has (rw- | r--) = (rw-) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the IPS bus is partitioned into two regions: one containing the MPU's programming model accessible only to the two processor cores and the remaining peripheral region accessible to both processors and the traditional DMA1 master.

This simple example is intended to show one possible application of the capabilities of the MPU in a typical system.

Chapter 14
Pin-Multiplexing and Control

14.1 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 14-1 is a block diagram of the device ports. Note that the GPIO functionality of the port IRQ pins is selected by the edge port modules. They are shown in the below figure only for completeness.

This chapter also includes registers for controlling the drive strengths and slew rates of the external pins.

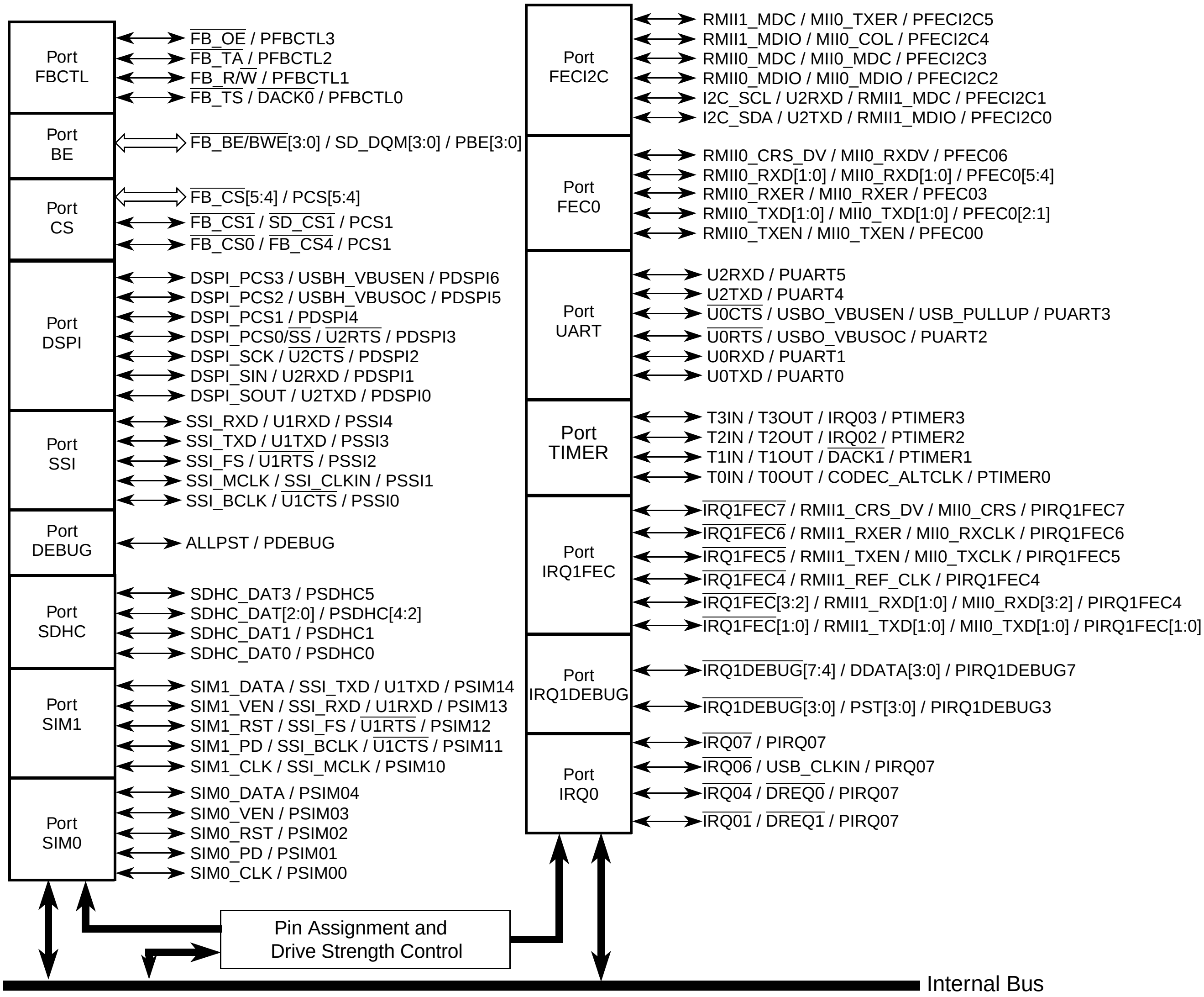

Port FBCTL
FB_OE / PFBCTL3
FB_TA / PFBCTL2
FB_R/W / PFBCTL1
FB_TS / DACK0 / PFBCTL0
Port BE
FB_BE/BWE[3:0] / SD_DQM[3:0] / PBE[3:0]
Port CS
FB_CS[5:4] / PCS[5:4]
FB_CS1 / SD_CS1 / PCS1
FB_CS0 / FB_CS4 / PCS1
Port DSPI
DSPI_PCS3 / USBH_VBUSEN / PDSPI6
DSPI_PCS2 / USBH_VBUSOC / PDSPI5
DSPI_PCS1 / PDSPI4
DSPI_PCS0/SS / U2RTS / PDSPI3
DSPI_SCK / U2CTS / PDSPI2
DSPI_SIN / U2RXD / PDSPI1
DSPI_SOUT / U2TXD / PDSPI0
Port SSI
SSI_RXD / U1RXD / PSSI4
SSI_TXD / U1TXD / PSSI3
SSI_FS / U1RTS / PSSI2
SSI_MCLK / SSI_CLKIN / PSSI1
SSI_BCLK / U1CTS / PSSI0
Port DEBUG
ALLPST / PDEBUG
Port SDHC
SDHC_DAT3 / PSDHC5
SDHC_DAT[2:0] / PSDHC[4:2]
SDHC_DAT1 / PSDHC1
SDHC_DAT0 / PSDHC0
Port SIM1
SIM1_DATA / SSI_TXD / U1TXD / PSIM14
SIM1_VEN / SSI_RXD / U1RXD / PSIM13
SIM1_RST / SSI_FS / U1RTS / PSIM12
SIM1_PD / SSI_BCLK / U1CTS / PSIM11
SIM1_CLK / SSI_MCLK / PSIM10
Port SIM0
SIM0_DATA / PSIM04
SIM0_VEN / PSIM03
SIM0_RST / PSIM02
SIM0_PD / PSIM01
SIM0_CLK / PSIM00
Port FECI2C
RMII1_MDC / MII0_TXER / PFECI2C5
RMII1_MDIO / MII0_COL / PFECI2C4
RMII0_MDC / MII0_MDC / PFECI2C3
RMII0_MDIO / MII0_MDIO / PFECI2C2
I2C_SCL / U2RXD / RMII1_MDC / PFECI2C1
I2C_SDA / U2TXD / RMII1_MDIO / PFECI2C0
Port FEC0
RMII0_CRS_DV / MII0_RXDV / PFEC06
RMII0_RXD[1:0] / MII0_RXD[1:0] / PFEC0[5:4]
RMII0_RXER / MII0_RXER / PFEC03
RMII0_TXD[1:0] / MII0_TXD[1:0] / PFEC0[2:1]
RMII0_TXEN / MII0_TXEN / PFEC00
Port UART
U2RXD / PUART5
U2TXD / PUART4
U0CTS / USBO_VBUSEN / USB_PULLUP / PUART3
U0RTS / USBO_VBUSOC / PUART2
U0RXD / PUART1
U0TXD / PUART0
Port TIMER
T3IN / T3OUT / IRQ03 / PTIMER3
T2IN / T2OUT / IRQ02 / PTIMER2
T1IN / T1OUT / DACK1 / PTIMER1
T0IN / T0OUT / CODEC_ALTCLK / PTIMER0
Port IRQ1FEC
IRQ1FEC7 / RMII1_CRS_DV / MII0_CRS / PIRQ1FEC7
IRQ1FEC6 / RMII1_RXER / MII0_RXCLK / PIRQ1FEC6
IRQ1FEC5 / RMII1_TXEN / MII0_TXCLK / PIRQ1FEC5
IRQ1FEC4 / RMII1_REF_CLK / PIRQ1FEC4
IRQ1FEC[3:2] / RMII1_RXD[1:0] / MII0_RXD[3:2] / PIRQ1FEC4
IRQ1FEC[1:0] / RMII1_TXD[1:0] / MII0_TXD[1:0] / PIRQ1FEC[1:0]
Port IRQ1DEBUG
IRQ1DEBUG[7:4] / DDATA[3:0] / PIRQ1DEBUG7
IRQ1DEBUG[3:0] / PST[3:0] / PIRQ1DEBUG3
Port IRQ0
IRQ07 / PIRQ07
IRQ06 / USB_CLKIN / PIRQ07
IRQ04 / DREQ0 / PIRQ07
IRQ01 / DREQ1 / PIRQ07
Pin Assignment and Drive Strength Control
Internal Bus

Figure 14-1. Pin Multiplexing and Control Block Diagram

14.1.1 Overview

The external pin-muxing and control module configures various external pins, including those used for:

- External bus accesses
- External device selection
- eSDHC
- SIM ports
- Ethernet data and control
- I^{2}C
- DSPI

- UART
- Edge ports
- 32-bit DMA timers

14.1.2 Features

The module includes these distinctive features:

- Control of primary function use
  - On all supported GPIO ports
  - On pins whose GPIO is not supported by the module: $\overline{\text{IRQ1}}$[7:1], $\overline{\text{IRQ0}}$[7,6,4,1]
- General purpose I/O support for all ports
  - Registers for storing output pin data
  - Registers for controlling pin data direction
  - Registers for reading current pin state
  - Registers for setting and clearing output pin data registers
- Control of functional pad drive strengths
- Slew rate control

14.2 External Signal Description

The external pins that are controllable by this module are listed in the below table under the GPIO column.

NOTE

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., FB_A23), while designations for multiple signals within a group use brackets (i.e., FB_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. Most pins that are muxed with GPIO will default to their GPIO functionality. See Table 14-1 for a list of the exceptions.

Table 14-1. Special-Case Default Signal Functionality

Pin	Default Signal
$\overline{\text{FB\_BE/BWE}}$[3:0]	$\overline{\text{FB\_BE/BWE}}$[3:0]
$\overline{\text{FB\_CS}}$[3:0]	$\overline{\text{FB\_CS}}$[3:0]
$\overline{\text{FB\_OE}}$	$\overline{\text{FB\_OE}}$
$\overline{\text{FB\_TA}}$	$\overline{\text{FB\_TA}}$

Table 14-1. Special-Case Default Signal Functionality (continued)

Pin	Default Signal
FB_R/$\overline{\text{W}}$	FB_R/$\overline{\text{W}}$
$\overline{\text{FB\_TS}}$	$\overline{\text{FB\_TS}}$

Table 14-2. MCF5301*x* Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
Reset								
$\overline{\text{RESET}}$	—	—	—	U	I	EVDD	41	M3
$\overline{\text{RSTOUT}}$	—	—	—	—	O	EVDD	42	N1
Clock								
EXTAL	—	—	—	—	I	EVDD	49	T2
XTAL	—	—	—	U[3]	O	EVDD	50	T3
Mode Selection								
BOOTMOD[1:0]	—	—	—	—	I	EVDD	55, 17	J5, G5
FlexBus								
FB_A[23:22]	—	$\overline{\text{FB\_CS}}$[3:2]	—	—	O	SDVDD	115, 114	P16, N16
FB_A[21:16]	—	—	—	—	O	SDVDD	113–108	R16, N14, N15, P15-13
FB_A[15:14]	—	SD_BA[1:0]	—	—	O	SDVDD	107, 106	R15, R14
FB_A[13:11]	—	SD_A[13:11]	—	—	O	SDVDD	105–103	N13, R12, R13
FB_A10	—	—	—	—	O	SDVDD	100	N12
FB_A[9:0]	—	SD_A[9:0]	—	—	O	SDVDD	99–97 95–89	P12, T14, T15, R11, P11, N11, T13, R10, T11, T12
FB_D[31:16]	—	SD_D[31:16]	—	—	I/O	SDVDD	208–198, 57–62, 64, 65	B3, A2, D6, C5, B4, A3, B5, C6, D12, C14, B14, C13, D11, B13, A14, A13
FB_D[15:0]	—	FB_D[31:16]	—	—	I/O	SDVDD	182–189, 177–170	B9, A9, A8, D7, B8, C8, D8, B7, C10, A10, B10, D10, C11, A11, B11, A12
FB_CLK	—	—	—	—	O	SDVDD	153	D13
$\overline{\text{FB\_BE/BWE}}$[3:0]	PBE[3:0]	SD_DQM[3:0]	—	—	O	SDVDD	197, 166, 179, 178	A4, B12, C9, D9
$\overline{\text{FB\_CS}}$[5:4]	PCS[5:4]	—	—	—	O	SDVDD	—	B6, C7
$\overline{\text{FB\_CS1}}$	PCS1	$\overline{\text{SD\_CS1}}$	—	—	O	SDVDD	5	D2
$\overline{\text{FB\_CS0}}$	PCS0	$\overline{\text{FB\_CS4}}$	—	—	O	SDVDD	6	C2

Table 14-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
FB_OE	PFBCTL3	—	—	—	O	SDVDD	1	D4
FB_TA	PFBCTL2	—	—	U	I	SDVDD	3	B2
FB_R/W	PFBCTL1	—	—	—	O	SDVDD	2	C3
FB_TS	PFBCTL0	DACK0	—	—	O	SDVDD	4	D3
SDRAM Controller								
SD_A10	—	—	—	—	O	SDVDD	206	C4
SD_CAS	—	—	—	—	O	SDVDD	154	D15
SD_CKE	—	—	—	—	O	SDVDD	151	B15
SD_CLK	—	—	—	—	O	SDVDD	190	A7
SD_CLK	—	—	—	—	O	SDVDD	191	A6
SD_CS0	—	—	—	—	O	SDVDD	155	A15
SD_DQS[1:0]	—	—	—	—	O	SDVDD	196, 167	C12, A5
SD_RAS	—	—	—	—	O	SDVDD	152	C15
SD_SDR_DQS	—	—	—	—	I	SDVDD	207	D5
SD_WE	—	—	—	—	O	SDVDD	150	D14
External Interrupts Port 1[4,5]								
IRQ1DEBUG[7:4]	PIRQ1DEBUG [7:4]	DDATA[3:0]	—	—	I	EVDD	—	H1, H4-2
IRQ1DEBUG[3:0]	PIRQ1DEBUG [3:0]	PST[3:0]	—	—	I	EVDD	—	K14, H14, K15, J13
IRQ1FEC7	PIRQ1FEC7	RMII1_CRS_DV	MII0_CRS	—	I	EVDD	29	J1
IRQ1FEC6	PIRQ1FEC6	RMII1_RXER	MII0_RXCLK	—	I	EVDD	30	J2
IRQ1FEC5	PIRQ1FEC5	RMII1_TXEN	MII0_TXCLK	—	I	EVDD	31	K4
IRQ1FEC4	PIRQ1FEC4	RMII1_REF_CLK	—	D	I	EVDD	32	J3
IRQ1FEC[3:2]	PIRQ1FEC[3:2]	RMII1_RXD[1:0]	MII0_RXD[3:2]	—	I	EVDD	33, 34	J4, K1
IRQ1FEC[1:0]	PIRQ1FEC[1:0]	RMII1_TXD[1:0]	MII0_TXD[3:2]	—	I	EVDD	35, 36	K2, L1
External Interrupts Port 0[5]								
IRQ07	PIRQ07	—	—	U	I	EVDD	10	E4
IRQ06	PIRQ06	—	USB_CLKIN	U	I	EVDD	—	L13
IRQ04	PIRQ04	DREQ0	—	U	I	EVDD	19	D1

Table 14-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
$\overline{\text{IRQ01}}$	PIRQ01	$\overline{\text{DREQ1}}$	—	U	I	EVDD	11	F4
Enhanced Secure Digital Host Controller								
SDHC_DAT3	PSDHC5	—	—	UD	I/O	EVDD	60	N4
SDHC_DAT[2:0]	PSDHC[4:2]	—	—	U	I/O	EVDD	61–63	R5, N6, N5
SDHC_CMD	PSDHC1	—	—	U	I/O	EVDD	59	R4
SDHC_CLK	PSDHC0	—	—	—	O	EVDD	58	R3
Codec								
CODEC_ADCN	—	AMP_MICN	—	—	I		85	P10
CODEC_ADCP	—	AMP_MICP	—	—	I		84	P9
CODEC_BGRVREF	—	—	—	—	I		86	N9
CODEC_DACN	—	AMP_HSN	—	—	O		75	R7
CODEC_DACP	—	AMP_HSP	—	—	O		67	R6
CODEC_REGBYP	—	—	—	—	I		81	P6
CODEC_REFN	—	—	—	—	I		79	P8
CODEC_REFP	—	—	—	—	I		78	P7
CODEC_VAG	—	—	—	—	I		82	N7
Amplifiers								
AMP_HPDUMMY	—	—	—	—	O		—	R9
AMP_HPOUT	—	—	—	—	O		—	R8
AMP_SPKRN	—	—	—	—	O		—	T9
AMP_SPKRP	—	—	—	—	O		—	T7
Smart Card interface 1								
SIM1_DATA	PSIM14	SSI_TXD	U1TXD	UD	I/O	EVDD	141	E14
SIM1_VEN	PSIM13	SSI_RXD	U1RXD	UD	O	EVDD	142	D16
SIM1_RST	PSIM12	SSI_FS	$\overline{\text{U1RTS}}$	—	O	EVDD	144	E13
SIM1_PD	PSIM11	SSI_BCLK	$\overline{\text{U1CTS}}$	—	O	EVDD	145	E15
SIM1_CLK	PSIM10	SSI_MCLK	—	—	O	EVDD	143	F13
Smart Card interface 0								
SIM0_DATA	PSIM04	—	—	—	I/O	EVDD	—	L3

Table 14-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
SIM0_VEN	PSIM03	—	—	—	O	EVDD	—	M2
SIM0_RST	PSIM02	—	—	—	O	EVDD	—	F16
SIM0_PD	PSIM01	—	—	—	O	EVDD	—	L14
SIM0_CLK	PSIM00	—	—	—	O	EVDD	—	M16
USB On-the-Go								
USBO_DM	—	—	—	—	O	USB VDD	148	C16
USBO_DP	—	—	—	—	O	USB VDD	149	B16
USB Host								
USBH_DM	—	—	—	—	O	USB VDD	—	B1
USBH_DP	—	—	—	—	O	USB VDD	—	C1
FEC 1								
RMII1_MDC	PFECI2C5	—	MII0_TXER	—		EVDD	22	E1
RMII1_MDIO	PFECI2C4	—	MII0_COL	—		EVDD	23	F1
FEC 0								
RMII0_CRS_DV	PFEC06	—	MII0_RXDV	—		EVDD	131	G16
RMII0_RXD[1:0]	PFEC0[5:4]	—	MII0_RXD[1:0]	—		EVDD	130, 129	H15, H16
RMII0_RXER	PFEC03	—	MII0_RXER	—		EVDD	127	J16
RMII0_TXD[1:0]	PFEC0[2:1]	—	MII0_TXD[1:0]	—		EVDD	125, 124	J15, J14
RMII0_TXEN	PFEC00	—	MII0_TXEN	D		EVDD	123	K16
RMII0_MDC	PFECI2C3	—	MII0_MDC	—		EVDD	133	G14
RMII0_MDIO	PFECI2C2	—	MII0_MDIO	—		EVDD	132	G15
Real Time Clock								
RTC_EXTAL	—	—	—	—	I	EVDD	—	P1
RTC_XTAL	—	—	—	—	O	EVDD	—	R1
Synchronous Serial Interface								
SSI_RXD	PSSI4	—	U1RXD	UD	I	EVDD	—	N3
SSI_TXD	PSSI3	—	U1TXD	UD	O	EVDD	—	P3

Table 14-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Pull-up (U)[1] Pull-down (D)	Direction[2]	Voltage Domain	MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP	MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA
SSI_FS	PSSI2	—	$\overline{\text{U1RTS}}$	—	I/O	EVDD	—	R2
SSI_MCLK	PSSI1	—	SSI_CLKIN	—	O	EVDD	—	P4
SSI_BCLK	PSSI0	—	$\overline{\text{U1CTS}}$	—	I/O	EVDD	—	P5
I^{2}C								
I2C_SCL	PFECI2C1	U2RXD	RMII1_MDC	U	I/O	EVDD	37	M1
I2C_SDA	PFECI2C0	U2TXD	RMII1_MDIO	U	I/O	EVDD	38	K3
DSPI								
DSPI_PCS3	PDSPI6	USBH_VBUS_EN	—	—	I/O	EVDD	—	P2
DSPI_PCS2	PDSPI5	USBH_VBUS_OC	—	—	I/O	EVDD	—	N2
DSPI_PCS1	PDSPI4	—	—	—	I/O	EVDD	140	F14
DSPI_PCS0/$\overline{\text{SS}}$	PDSPI3	$\overline{\text{U2RTS}}$	—	U	I/O	EVDD	137	G13
DSPI_SCK	PDSPI2	$\overline{\text{U2CTS}}$	—	—	I/O	EVDD	134	H13
DSPI_SIN	PDSPI1	U2RXD	—	—	I	EVDD	136	E16
DSPI_SOUT	PDSPI0	U2TXD	—	—	O	EVDD	135	F15
UARTs								
U2RXD	PUART5	—	—	—	I	EVDD	14	E2
U2TXD	PUART4	—	—	—	O	EVDD	18	F2
$\overline{\text{U0CTS}}$	PUART3	USBO_VBUS_EN	USB_PULLUP	—	I	EVDD	20	G4
$\overline{\text{U0RTS}}$	PUART2	USBO_VBUS_OC	—	—	O	EVDD	21	G3
U0RXD	PUART1	—	—	—	I	EVDD	27	G2
U0TXD	PUART0	—	—	—	O	EVDD	28	G1
DMA Timers								
T3IN	PTIMER3	T3OUT	IRQ03	—	I	EVDD	13	F3
T2IN	PTIMER2	T2OUT	IRQ02	—	I	EVDD	12	E3
T1IN	PTIMER1	T1OUT	$\overline{\text{DACK1}}$	—	I	EVDD	122	K13
T0IN	PTIMER0	T0OUT	CODEC_ALTCLK	—	I	EVDD	121	L16
BDM/JTAG[6]								
ALLPST	PDEBUG	—	—	—	O	EVDD	43	—
JTAG_EN	—	—	—	D	I	EVDD	64	M8

Table 14-2. MCF5301*x* Signal Information and Muxing (continued)

Signal Name	**GPIO**	**Alternate 1**	**Alternate 2**	**Pull-up (U)[1] Pull-down (D)**	**Direction[2]**	**Voltage Domain**	**MCF53010 MCF53011 MCF53012 MCF53013 208 LQFP**	**MCF53014 MCF53015 MCF53016 MCF53017 256 MAPBGA**
PSTCLK	—	TCLK	—	—	I	EVDD	65	T5
DSI	—	TDI	—	U	I	EVDD	66	T4
DSO	—	TDO	—	—	O	EVDD	120	M15
$\overline{\text{BKPT}}$	—	TMS	—	U	I	EVDD	119	M14
DSCLK	—	$\overline{\text{TRST}}$	—	U	I	EVDD	118	L15
Test								
TEST	—	—	—	D	I	EVDD	146	F12
Power Supplies								
IVDD	—	—	—	—	—	—	16, 44, 69, 77, 128, 169, 193	E9, F8, F9, H5, H6, H11, H12, J6, J11, L8, L9
EVDD	—	—	—	—	—	—	9, 24, 26, 40, 47, 51, 54, 57, 74, 126, 139, 195	F5, G6, G11, G12, J12, K6, K11, K12, L5-7, L10-12, M5-7, M12
SD_VDD	—	—	—	—	—	—	7, 102, 116, 156, 163, 181, 208	E5, E6, E10-12, F6, F7, F10, F11
VDD_OSC_A_PLL	—	—	—	—	—	—	46	M4
VDD_USBO	—	—	—	—	—	—	147	E7
VDD_USBH	—	—	—	—	—	—	—	E8
VDD_RTC	—	—	—	—	—	—	—	
AVDD_CODEC	—	—	—	—	—	—	80	N8
AVDD_SPKR	—	—	—	—	—	—	—	T8
VDD_EPM	—	—	—	—	—	—	96	M9
VSTBY_SRAM	—	—	—	—	—	—	—	L2
VSTBY_RTC	—	—	—	—	—	—	—	L4
VSS	—	—	—	—	—	—	8, 15, 25, 39, 45, 48, 52, 53, 56, 68, 73, 76, 101, 117, 138, 168, 180, 192, 194	A1, A16, G7-10, H7-10, J7-10, K7-10, T1, T16
VSS_CODEC	—	—	—	—	—	—	83	N10
AVSS_SPKR_HDST	—	—	—	—	—	—	—	T6
AVSS_SPKR_HP	—	—	—	—	—	—	—	T10

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

[2] Refers to pin’s primary function.

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

[4] The edge port 1 signals are the primary functions on two sets of pins (IRQ1FEC*n* and IRQ1DEBUG*n*). If an IRQ1 function is configured on both pins, the IRQ1FEC*n* pin takes priority. The corresponding IRQ1DEBUG*n* pin is disconnected internally from the edge port 1 module.

[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.

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

As shown in Table 14-2, 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 should be avoided for input functions to prevent unexpected behavior. All multiple-pin functions are listed in Table 14-3.

signal

Table 14-3. Multiple-Pin Functions

Function	Direction	Associated Pins
$\overline{\text{U1CTS}}$	I	SIM1_PD, SSI_BCLK
$\overline{\text{U1RTS}}$	O	SIM1_RST, SSI_FS
U1RXD	I	SIM1_VEN, SSI_RXD
U1TXD	O	SIM1_DATA, SSI_TXD
U2RXD	I	I2C_SCL, DSPI_SIN, U2RXD
U2TXD	O	I2C_SDA, DSPI_SOUT, U2TXD
IRQ1[7:0][1]	I	IRQ1FEC[7:0], IRQ1DEBUG[7:0]

[1] Since the edge port signals are the primary function of the IRQ1FEC*n* and IRQ1DEBUG*n* pins, the processor contains priority logic. If both sets of pins are configured as edge port, the IRQ1FEC*n* pins function as edge port signals and the IRQ1DEBUG*n* pins are disconnected from the edge port module.

14.3 Memory Map/Register Definition

Table 14-4 summarizes all the registers in the pin multiplexing and control address space.

Table 14-4. Pin Multiplexing and Control Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Port Output Data Registers					
0xFC0A_4000	PODR_FBCTL	8	R/W	0x0F	14.3.1/14-13
0xFC0A_4001	PODR_BE	8	R/W	0x0F	14.3.1/14-13
0xFC0A_4002	PODR_CS	8	R/W	0x33	14.3.1/14-13
0xFC0A_4003	PODR_DSPI	8	R/W	0x7F	14.3.1/14-13
0xFC0A_4005	PODR_FEC0	8	R/W	0x7F	14.3.1/14-13
0xFC0A_4006	PODR_FECI2C	8	R/W	0x3F	14.3.1/14-13
0xFC0A_4009	PODR_SIMP1	8	R/W	0x1F	14.3.1/14-13

Table 14-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_400A	PODR_SIMP0	8	R/W	0x1F	14.3.1/14-13
0xFC0A_400B	PODR_TIMER	8	R/W	0x0F	14.3.1/14-13
0xFC0A_400C	PODR_UART	8	R/W	0x3F	14.3.1/14-13
0xFC0A_400D	PODR_DEBUG	8	R/W	0x01	14.3.1/14-13
0xFC0A_400F	PODR_SDHC	8	R/W	0x3F	14.3.1/14-13
0xFC0A_4010	PODR_SSI	8	R/W	0x1F	14.3.1/14-13
Port Data Direction Registers					
0xFC0A_4014	PDDR_FBCTL	8	R/W	0xFF	14.3.2/14-15
0xFC0A_4015	PDDR_BE	8	R/W	0xFF	14.3.2/14-15
0xFC0A_4016	PDDR_CS	8	R/W	0xFF	14.3.2/14-15
0xFC0A_4017	PDDR_DSPI	8	R/W	0xFF	14.3.2/14-15
0xFC0A_4019	PDDR_FEC0	8	R/W	0x00	14.3.2/14-15
0xFC0A_401A	PDDR_FECI2C	8	R/W	0x00	14.3.2/14-15
0xFC0A_401D	PDDR_SIMP1	8	R/W	0x00	14.3.2/14-15
0xFC0A_401E	PDDR_SIMP0	8	R/W	0x00	14.3.2/14-15
0xFC0A_401F	PDDR_TIMER	8	R/W	0x00	14.3.2/14-15
0xFC0A_4020	PDDR_UART	8	R/W	0x00	14.3.2/14-15
0xFC0A_4021	PDDR_DEBUG	8	R/W	0x00	14.3.2/14-15
0xFC0A_4023	PDDR_SDHC	8	R/W	0x00	14.3.2/14-15
0xFC0A_4024	PDDR_SSI	8	R/W	0x00	14.3.2/14-15
Port Pin Data/Set Data Registers					
0xFC0A_4028	PPDSDR_FBCTL	8	R/W	See Section	14.3.3/14-17
0xFC0A_4029	PPDSDR_BE	8	R/W	See Section	14.3.3/14-17
0xFC0A_402A	PPDSDR_CS	8	R/W	See Section	14.3.3/14-17
0xFC0A_402B	PPDSDR_DSPI	8	R/W	See Section	14.3.3/14-17
0xFC0A_402D	PPDSDR_FEC0	8	R/W	See Section	14.3.3/14-17
0xFC0A_402E	PPDSDR_FECI2C	8	R/W	See Section	14.3.3/14-17
0xFC0A_4031	PPDSDR_SIMP1	8	R/W	See Section	14.3.3/14-17
0xFC0A_4032	PPDSDR_SIMP0	8	R/W	See Section	14.3.3/14-17
0xFC0A_4033	PPDSDR_TIMER	8	R/W	See Section	14.3.3/14-17
0xFC0A_4034	PPDSDR_UART	8	R/W	See Section	14.3.3/14-17
0xFC0A_4035	PPDSDR_DEBUG	8	R/W	See Section	14.3.3/14-17

Table 14-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4037	PPDSDR_SDHC	8	R/W	See Section	14.3.3/14-17
0xFC0A_4038	PPDSDR_SSI	8	R/W	See Section	14.3.3/14-17
Port Clear Output Data Registers					
0xFC0A_403C	PCLRR_FBCTL	8	W	0x00	14.3.4/14-19
0xFC0A_403D	PCLRR_BE	8	W	0x00	14.3.4/14-19
0xFC0A_403E	PCLRR_CS	8	W	0x00	14.3.4/14-19
0xFC0A_403F	PCLRR_DSPI	8	W	0x00	14.3.4/14-19
0xFC0A_4041	PCLRR_FEC0	8	W	0x00	14.3.4/14-19
0xFC0A_4042	PCLRR_FECI2C	8	W	0x00	14.3.4/14-19
0xFC0A_4045	PCLRR_SIMP1	8	W	0x00	14.3.4/14-19
0xFC0A_4046	PCLRR_SIMP0	8	W	0x00	14.3.4/14-19
0xFC0A_4047	PCLRR_TIMER	8	W	0x00	14.3.4/14-19
0xFC0A_4048	PCLRR_UART	8	W	0x00	14.3.4/14-19
0xFC0A_4049	PCLRR_DEBUG	8	W	0x00	14.3.4/14-19
0xFC0A_404B	PCLRR_SDHC	8	W	0x00	14.3.4/14-19
0xFC0A_404C	PCLRR_SSI	8	W	0x00	14.3.4/14-19
Pin Assignment Registers					
0xFC0A_4050	PAR_FBCTL	8	R/W	0xF8	14.3.5.1/14-21
0xFC0A_4051	PAR_BE	8	R/W	0x55	14.3.5.2/14-21
0xFC0A_4052	PAR_CS	8	R/W	See Section	14.3.5.3/14-22
0xFC0A_4054	PAR_DSPIH	8	R/W	0x00	14.3.5.4/14-23
0xFC0A_4055	PAR_DSPIL	8	R/W	0x00	14.3.5.4/14-23
0xFC0A_4056	PAR_FEC	8	R/W	0x00	14.3.5.5/14-24
0xFC0A_4057	PAR_FECI2C	8	R/W	0x00	14.3.5.5/14-24
0xFC0A_4058	PAR_IRQ0H	8	R/W	0x00	14.3.5.6/14-27
0xFC0A_4059	PAR_IRQ0L	8	R/W	0x00	14.3.5.6/14-27
0xFC0A_405A	PAR_IRQ1DEBUGH	8	R/W	0x00	14.3.5.7/14-28
0xFC0A_405B	PAR_IRQ1DEBUGL	8	R/W	0x55	14.3.5.7/14-28
0xFC0A_405C	PAR_SIMP1H	8	R/W	0x00	14.3.5.8/14-30
0xFC0A_405D	PAR_SIMP1L	8	R/W	0x00	14.3.5.8/14-30
0xFC0A_405E	PAR_SIMP0	8	R/W	0x00	14.3.5.9/14-31
0xFC0A_405F	PAR_TIMER	8	R/W	0x00	14.3.5.10/14-31

Table 14-4. Pin Multiplexing and Control Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4060	PAR_UART	8	R/W	0x00	14.3.5.11/14-32
0xFC0A_4062	PAR_DEBUG	8	R/W	0x80	14.3.5.12/14-33
0xFC0A_4063	PAR_SDHC	8	R/W	0x00	14.3.5.13/14-33
0xFC0A_4064	PAR_SSIH	8	R/W	0x00	14.3.5.14/14-34
0xFC0A_4065	PAR_SSIL	8	R/W	0x00	14.3.5.14/14-34
Mode Select Control Registers					
0xFC0A_4068	MSCR_1	8	R/W	0xC0	14.3.6/14-35
0xFC0A_4069	MSCR_2	8	R/W	0xC0	14.3.6/14-35
0xFC0A_4070	MSCR_3	8	R/W	0xC0	14.3.6/14-35
0xFC0A_4071	MSCR_4	8	R/W	0xD8	14.3.6/14-35
Slew Rate/Drive Strength Control Registers					
0xFC0A_406C	SRCR_DSPI	8	R/W	See Section	14.3.7/14-36
0xFC0A_406D	DSCR_FEC	8	R/W	See Section	14.3.8/14-38
0xFC0A_406E	SRCR_I2C	8	R/W	See Section	14.3.7/14-36
0xFC0A_406F	SRCR_IRQ	8	R/W	See Section	14.3.7/14-36
0xFC0A_4070	SRCR_SIM	8	R/W	See Section	14.3.7/14-36
0xFC0A_4071	SRCR_TIMER	8	R/W	See Section	14.3.7/14-36
0xFC0A_4072	SRCR_UART	8	R/W	See Section	14.3.7/14-36
0xFC0A_4074	SRCR_SDHC	8	R/W	See Section	14.3.7/14-36
0xFC0A_4075	SRCR_SSI	8	R/W	See Section	14.3.7/14-36
Pull Control Registers					
0xFC0A_4078	PCRH	8	R/W	0xFB	14.3.9/14-39
0xFC0A_4079	PCRL	8	R/W	0xFC	14.3.9/14-39

14.3.1 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 PPDSDR_*x*. To clear bits in a PODR_*x* register, clear the PODR_*x* bits, or clear the corresponding bits in PCLRR_*x*.

Address: 0xFC0A_4003 (PODR_DSPI)
0xFC0A_4005 (PODR_FEC0)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PODR_*x*						
W								
Reset:	0	1	1	1	1	1	1	1

Figure 14-2. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4006 (PODR_FECI2C)
0xFC0A_400C (PODR_UART)
0xFC0A_400F (PODR_SDHC)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PODR_*x*					
W								
Reset:	0	0	1	1	1	1	1	1

Figure 14-3. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4009 (PODR_SIMP1)
0xFC0A_400A (PODR_SIMP0)
0xFC0A_4010 (PODR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PODR_*x*				
W								
Reset:	0	0	0	1	1	1	1	1

Figure 14-4. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4000 (PODR_FBCTL)
0xFC0A_4001 (PODR_BE)
0xFC0A_400B (PODR_TIMER)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PODR_*x*			
W								
Reset:	0	0	0	0	1	1	1	1

Figure 14-5. Port *x* Output Data Registers (PODR_*x*)

Address: 0xFC0A_4002 (PODR_CS)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PODR_CS[3:2]		0	0	PODR_CS[1:0]	
W								
Reset:	0	0	1	1	0	0	1	1

Figure 14-6. Port CS Output Data Registers (PODR_CS)

Address: 0xFC0A_400D (PODR_DEBUG)

Access: User read/write

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

Figure 14-7. Port Debug Output Data Registers (PODR_DEBUG)

Table 14-5. PODR_*x* Field Descriptions

Field	Description
PODR_*x*	Port *x* output data bits. 0 Drives 0 when the port *x* pin is general purpose output 1 Drives 1 when the port *x* pin is general purpose output

Note: See above figures for bit field positions.

14.3.2 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_DSPI)
0xFC0A_4019 (PDDR_FEC0)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PDDR_*x*						
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-8. Port *x* Data Direction Registers (PDDR_*x*)

Address: 0xFC0A_401A (PDDR_FECI2C)
0xFC0A_4020 (PDDR_UART)
0xFC0A_4023 (PDDR_SDHC)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PDDR_*x*					
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-9. Port *x* Data Direction Registers (PDDR_*x*)

Address: 0xFC0A_401D (PDDR_SIMP1)
0xFC0A_401E (PDDR_SIMP0)
0xFC0A_4024 (PDDR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PDDR_*x*				
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-10. Port *x* Output Data Registers (PDDR_*x*)

Address: 0xFC0A_4014 (PDDR_FBCTL)
0xFC0A_4015 (PDDR_BE)
0xFC0A_401F (PDDR_TIMER)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PDDR_*x*			
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-11. Port *x* Output Data Registers (PDDR_*x*)

Address: 0xFC0A_4016 (PDDR_CS)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PDDR_CS[3:2]		0	0	PDDR_CS[1:0]	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-12. Port CS Output Data Registers (PDDR_CS)

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

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

Figure 14-13. Port DEBUG Output Data Registers (PDDR_DEBUG)

Table 14-6. PDDR_*x* Field Descriptions

Field	Description
PDDR_*x*	Port *x* output data direction bits. 1 Port *x* pin configured as output 0 Port *x* pin configured as input

Note: See above figures for bit field positions.

14.3.3 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_DSPI) Access: User read/write
0xFC0A_402D (PPDSDR_FEC0)

	7	6	5	4	3	2	1	0
R	0	PPDR_*x*						
W		PSDR_*x*						
Reset:	0	[P*x*6]	[P*x*5]	[P*x*4]	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 14-14. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_402E (PPDSDR_FECI2C) Access: User read/write
0xFC0A_4034 (PPDSDR_UART)
0xFC0A_4037 (PPDSDR_SDHC)

	7	6	5	4	3	2	1	0
R	0	0	PPDR_*x*					
W			PSDR_*x*					
Reset:	0	0	[P*x*5]	[P*x*4]	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 14-15. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_4031 (PPDSDR_SIMP1)
0xFC0A_4032 (PPDSDR_SIMP0)
0xFC0A_4038 (PPDSDR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	PPDR_*x*				
W				PSDR_*x*				
Reset:	0	0	0	[P*x*4]	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 14-16. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*)

Address: 0xFC0A_4028 (PPDSDR_FBCTL)
0xFC0A_4029 (PPDSDR_BE)
0xFC0A_4033 (PPDSDR_TIMER)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PPDR_*x*			
W					PSDR_*x*			
Reset:	0	0	0	0	[P*x*3]	[P*x*2]	[P*x*1]	[P*x*0]

Figure 14-17. Port *x* Pin Data/Set Data Registers (PPDSDR_*x*

Address: 0xFC0A_402A (PPDSDR_CS)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PPDR_CS[3:2]		0	0	PPDR_CS[1:0]	
W			PSDR_CS[3:2]				PSDR_CS[1:0]	
Reset:	0	0	[PCS3]	[PCS2]	0	0	[PCS1]	[PCS0]

Figure 14-18. Port CS Pin Data/Set Data Registers (PPDSDR_CS)

Address: 0xFC0A_4035 (PPDSDR_DEBUG)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	PPDR_ DEBUG
W								PSDR_ DEBUG
Reset:	0	0	0	0	0	0	0	[PDEBUG]

Figure 14-19. Port DEBUG Pin Data/Set Data Registers (PPDSDR_DEBUG)

Table 14-7. PPDSDR_*x* Field Descriptions

Field	Description
PPDR_*x* (read)	Port *x* pin data bits. 0 Port *x* pin state is 0 1 Port *x* pin state is 1
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.

14.3.4 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_4039 (PCLRR_DSPI)
0xFC0A_4041 (PCLRR_FEC0)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		PCLRR_*x*						
Reset:	0	0	0	0	0	0	0	0

Figure 14-20. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_4042 (PCLRR_FECI2C)
0xFC0A_4048 (PCLRR_UART)
0xFC0A_404D (PCLRR_SDHC)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W			PCLRR_*x*					
Reset:	0	0	0	0	0	0	0	0

Figure 14-21. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_4045 (PCLRR_SIMP1)
0xFC0A_4046 (PCLRR_SIMP0)
0xFC0A_404C (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_*x*				
Reset:	0	0	0	0	0	0	0	0

Figure 14-22. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_403C (PCLRR_FBCTL)
0xFC0A_403D (PCLRR_BE)
0xFC0A_4047 (PCLRR_TIMER)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W					PCLRR_*x*			
Reset:	0	0	0	0	0	0	0	0

Figure 14-23. Port *x* Clear Output Data Registers (PCLRR_*x*)

Address: 0xFC0A_403E (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[3:2]				PCLRR_CS[1:0]	
Reset:	0	0	0	0	0	0	0	0

Figure 14-24. Port CS Clear Output Data Registers (PCLRR_CS)

Address: 0xFC0A_40 (PCLRR_DEBUG)

Access: User write-only

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

Figure 14-25. Port Debug Clear Output Data Registers (PCLRR_DEBUG)

Table 14-8. PCLRR_*x* Field Descriptions

Field	Description
PCLRR_*x*	Port *x* clear data bits. 0 Clears corresponding PODR_*x* bit 1 No effect

Note: See above figures for bit field positions.

14.3.5 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. The bit fields in the PAR_*x* registers are one or two bits wide. The encodings are shown in the tables below and correspond to the headings in Table 14-2.

Table 14-9. PAR_x Settings (Two Bit Field)

Value	Function
00	GPIO
01	Alternate 2 Function
10	Alternate 1 Function
11	Primary Function

Table 14-10. PAR_x Settings (One Bit Field)

Value	Function
0	GPIO
1	Primary Function

14.3.5.1 FlexBus Control Pin Assignment Register (PAR_FBCTL)

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

	7	6	5	4	3	2	1	0
R	PAR_OE	PAR_TA	PAR_RWB	PAR_TS		0	0	0
W								
Reset:	1	1	1	1	1	0	0	0

Figure 14-26. FlexBus Control Pin Assignment Register (PAR_FBCTL)

Table 14-11. PAR_FBCTL Field Descriptions

Field	Description
7 PAR_OE	0 GPIO 1 $\overline{\text{FB\_OE}}$
6 PAR_TA	0 GPIO 1 $\overline{\text{FB\_TA}}$
5 PAR_RWB	0 GPIO 1 FB_R/$\overline{\text{W}}$
4–3 PAR_TS	00 GPIO 01 Reserved 10 Reserved 11 $\overline{\text{FB\_TS}}$
2–0	Reserved, must be cleared.

14.3.5.2 Byte Enables Pin Assignment Register (PAR_BE)

Address: 0xFC0A_4051 (PAR_BE) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PAR_BE3	0	PAR_BE2	0	PAR_BE1	0	PAR_BE0
W								
Reset:	0	1	0	1	0	1	0	1

Figure 14-27. Byte Enable Pin Assignment Register (PAR_BE)

Table 14-12. PAR_BE Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 PAR_BE3	0 GPIO 1 FB_BE/BWE3 or SD_DQM3
5	Reserved, must be cleared.
4 PAR_BE2	0 GPIO 1 FB_BE/BWE2 or SD_DQM2
3	Reserved, must be cleared.
2 PAR_BE1	0 GPIO 1 FB_BE/BWE1 or SD_DQM1
1	Reserved, must be cleared.
0 PAR_BE0	0 GPIO 1 FB_BE/BWE0 or SD_DQM0

14.3.5.3 Chip Selects Pin Assignment Register (PAR_CS)

Address: 0xFC0A_4052 (PAR_CS) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PAR_CS5	0	PAR_CS4	PAR_CS1		PAR_CS0	
W								
Reset:	0	1	0	1	1	1	1	1

Figure 14-28. Chip Select Pin Assignment Register (PAR_CS)

Table 14-13. PAR_CS Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 PAR_CS5	0 GPIO 1 FB_CS5
5	Reserved, must be cleared.

Table 14-13. PAR_CS Field Descriptions (continued)

Field	Description
4 PAR_CS4	0 GPIO 1 $\overline{\text{FB\_CS4}}$
3–2 PAR_CS1	00 GPIO 01 Reserved 10 $\overline{\text{SD\_CS1}}$ 11 $\overline{\text{FB\_CS1}}$
1–0 PAR_CS0	00 GPIO 01 Reserved 10 $\overline{\text{FB\_CS4}}$ 11 $\overline{\text{FB\_CS0}}$

14.3.5.4 DSPI Pin Assignment Registers (PAR_DSPIH & PAR_DSPIL)

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

	7	6	5	4	3	2	1	0
R W	PAR_SIN		PAR_SOUT		PAR_SCK		PAR_PCS0	
Reset:	0	0	0	0	0	0	0	0

Figure 14-29. DSPI High Pin Assignment Register (PAR_DSPIH)

Table 14-14. PAR_DSPIH Field Descriptions

Field	Description
7–6 PAR_SIN	00 GPIO 01 Reserved 10 U2RXD 11 DSPI_SIN
5–4 PAR_SOUT	00 GPIO 01 Reserved 10 U2TXD 11 DSPI_SOUT
3–2 PAR_SCK	00 GPIO 01 Reserved 10 $\overline{\text{U2CTS}}$ 11 DSPI_SCK
1–0 PAR_PCS0	00 GPIO 01 Reserved 10 $\overline{\text{U2RTS}}$ 11 DSPI_PCS0

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

	7	6	5	4	3	2	1	0
R	PAR_PCS1		PAR_PCS2		PAR_PCS3		0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-30. DSPI Low Pin Assignment Register (PAR_DSPIL)

Table 14-15. PAR_DSPIL Field Descriptions

Field	Description
7–6 PAR_PCS1	00 GPIO 01 Reserved 10 Reserved 11 DSPI_PCS1
5–4 PAR_PCS2	00 GPIO 01 Reserved 10 USBH_VBUS_OC 11 DSPI_PCS2
3–2 PAR_PCS3	00 GPIO 01 Reserved 10 USBH_VBUS_EN 11 DSPI_PCS3
1–0	Reserved, must be cleared.

14.3.5.5 FEC Pin Assignment Registers (PAR_FEC & PAR_FECI2C)

The PAR_FEC and PAR_FECI2C registers (along with the CCM’s MISCCR register) configure the device’s pins for the various FEC functions (as shown in the below table), edge port 1, or GPIO functions. The listed settings do not configure the PHY management interface (MDC, MDIO) pins for functioning as mangement interface signals.

See Section 14.3.5.7, “Edge Port 1 Pin Assignment Registers (PAR_IRQ1DEBUGH & PAR_IRQ1DEBUGL)” for the pin assignment registers for the edge port 1 pins that are shared with the debug module.

Table 14-16. PAR_FEC and PAR_FECI2C Settings for FEC Modes

Function	PAR_FEC	PAR_FECI2C	CCM’s MISCCR

RMII1	PAR_RMII1 = 1
	PAR_7WIRE1 = 1
		—	FECM = 1

RMII0	PAR_RMII0 = 1
	PAR_7WIRE0 = 1
		—	FECM = 1

Table 14-16. PAR_FEC and PAR_FECI2C Settings for FEC Modes (continued)

Function	PAR_FEC	PAR_FECI2C	CCM’s MISCCR

7 Wire	PAR_7WIRE1 = 1
	PAR_7WIRE0 = 1
		PAR_RMII1_MDIO = 1	FECM = 0

MII	PAR_RMII1 = 1
	PAR_7WIRE1 = 1
	PAR_RMII0 = 1
	PAR_7WIRE0 = 1
		PAR_RMII1_MDC = 1
		PAR_RMII1_MDIO = 1
			FECM = 0

NOTE

RMII_REF_CLK is not directly controlled through PAR_FEC. Instead it is enabled when RMII1 or RMII0 functionality is enabled as per the above table. If neither RMII1 nor RMII0 is enabled, then this pin functions as IRQ1_4.

Address: 0xFC0A_4056 (PAR_FEC) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PAR_ 7WIRE1	0	PAR_ RMII1	0	PAR_ 7WIRE0	0	PAR_ RMII0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-31. FEC Pin Assignment (PAR_FEC)

Table 14-17. PAR_FEC Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be cleared.</td></tr>
<tr><td>6
PAR_7WIRE1</td><td><table><tr><th></th><th>IRQ16</th><th>IRQ15</th></tr><tr><td>0</td><td>IRQ16/GPIO</td><td>IRQ15/GPIO</td></tr><tr><td>1</td><td>RMII1_RXER</td><td>RMII1_TXEN</td></tr></table></td></tr>
<tr><td>5</td><td>Reserved, must be cleared.</td></tr>
<tr><td>4
PAR_RMII1</td><td><table><tr><th></th><th>IRQ17</th><th>IRQ1[3:2]</th><th>IRQ1[1:0]</th></tr><tr><td>0</td><td>IRQ17/GPIO</td><td>IRQ1[3:2]/GPIO</td><td>IRQ1[1:0]/GPIO</td></tr><tr><td>1</td><td>RMII1_CRS_DV</td><td>RMII1_RXD[1:0]</td><td>RMII1_TXD[1:0]</td></tr></table></td></tr>
<tr><td>3</td><td>Reserved, must be cleared.</td></tr>
</table>

Table 14-17. PAR_FEC Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
PAR_7WIRE0</td><td><table><tr><th></th><th>IRQ17</th><th>IRQ15</th><th>IRQ12</th><th>IRQ10</th></tr><tr><td>0</td><td>IRQ17/GPIO</td><td>IRQ15/GPIO</td><td>IRQ12/GPIO</td><td>IRQ10/GPIO</td></tr><tr><td>1</td><td>RMII1_CRS_DV</td><td>RMII1_TXEN</td><td>RMII0_RXD0</td><td>RMII0_TXD0</td></tr></table></td></tr>
<tr><td>1</td><td>Reserved, must be cleared.</td></tr>
<tr><td>0
PAR_RMII0</td><td><table><tr><th></th><th>IRQ16</th><th>IRQ13</th><th>IRQ11</th></tr><tr><td>0</td><td>IRQ16/GPIO</td><td>IRQ13/GPIO</td><td>IRQ11/GPIO</td></tr><tr><td>1</td><td>RMII1_RXER</td><td>RMII0_RXD1</td><td>RMII0_TXD1</td></tr></table></td></tr>
</table>

Address: 0xFC0A_4057 (PAR_FECI2C) Access: User read/write

	7	6	5	4	3	2	1	0
R W	PAR_RMII0_MDC	PAR_RMII0_MDIO	PAR_RMII1_MDC	PAR_RMII1_MDIO	PAR_SDA		PAR_SCL	
Reset:	0	0	0	0	0	0	0	0

Figure 14-32. FEC/I2C Pin Assignment (PAR_FECI2C)

Table 14-18. PAR_FECI2C Field Descriptions

Field	Description
7 PAR_RMII0_MDC	0 GPIO 1 RMII0_MDC
6 PAR_RMII0_MDIO	0 GPIO 1 RMII0_MDIO
5 PAR_RMII1_MDC	0 GPIO 1 RMII1_MDC

Table 14-18. PAR_FECI2C Field Descriptions (continued)

Field	Description
4 PAR_RMII1_MDIO	0 GPIO 1 RMII1_MDIO
3–2 PAR_SDA 1–0 PAR_SCL	I2C_SDA and I2C_SCL pin assignment. These bit fields configure the I2C_SCL and I2C_SDA pins for one of their primary functions or GPIO.

	PAR_SDA	PAR_SCL
00	GPIO	GPIO
01	RMII1_MDIO	RMII1_MDC
10	U2TXD	U2RXD
11	I2C_SDA	I2C_SCL

14.3.5.6 Edge Port 0 Pin Assignment Registers (PAR_IRQ0H & PAR_IRQ0L)

The PAR_IRQ0H/L registers control the functions of the edge port 0 pins. Selecting GPIO or edge port functionality is controlled by the edge port modules. See Chapter 16, “Edge Port Modules (EPORTn),” for details.

Address: 0xFC0A_4058 (PAR_IRQ0H) Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_IRQ07		PAR_IRQ06		0	0	PAR_IRQ04	
W					▨	▨		
Reset:	0	0	0	0	0	0	0	0

Figure 14-33. IRQ0 Pin Assignment High (PAR_IRQ0H)

Table 14-19. PAR_IRQ0H Field Descriptions

Field	Description
7–6 PAR_IRQ07 5–4 PAR_IRQ06	

	PAR_IRQ07	PAR_IRQ06
00	$\overline{\text{IRQ07}}$/GPIO	$\overline{\text{IRQ06}}$/GPIO
01	Reserved	$\overline{\text{USB\_CLKIN}}$
10	Reserved	Reserved
11	Reserved	Reserved

Table 14-19. PAR_IRQ0H Field Descriptions (continued)

Field	Description
3–2	Reserved, must be cleared.
1–0 PAR_IRQ04	00 $\overline{\text{IRQ04}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 01 Reserved 10 $\overline{\text{DREQ0}}$ 11 Reserved

Address: 0xFC0A_4059 (PAR_IRQ0L) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	PAR_IRQ01		0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-34. IRQ0 Pin Assignment Low (PAR_IRQ0L)

Table 14-20. PAR_IRQ0L Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3–2 PAR_IRQ01	00 $\overline{\text{IRQ01}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 01 Reserved 10 $\overline{\text{DREQ1}}$ 11 Reserved
1–0	Reserved, must be cleared.

14.3.5.7 Edge Port 1 Pin Assignment Registers (PAR_IRQ1DEBUGH & PAR_IRQ1DEBUGL)

The PAR_IRQ1DEBUGH/L registers control the functions of the edge port 1 pins that are shared with the debug signals. (See Section 14.3.5.5, “FEC Pin Assignment Registers (PAR_FEC & PAR_FECI2C)” for the pin assignment registers for the edge port 1 pins that are shared with the FEC module.)

Selecting GPIO or edge port functionality is controlled by the edge port modules. See Chapter 16, “Edge Port Modules (EPORTn),” for details.

Address: 0xFC0A_405A (PAR_IRQ1DEBUGH) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PAR_IRQ17	0	PAR_IRQ16	0	PAR_IRQ15	0	PAR_IRQ14
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-35. IRQ1 Pin Assignment High (PAR_IRQ1DEBUGH)

Table 14-21. PAR_IRQ1DEBUGH Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 PAR_IRQ17	0 $\overline{\text{IRQ17}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 DDATA3
5	Reserved, must be cleared.
4 PAR_IRQ16	0 $\overline{\text{IRQ16}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 DDATA2
1	Reserved, must be cleared.
2 PAR_IRQ15	0 $\overline{\text{IRQ15}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 DDATA1
1	Reserved, must be cleared.
0 PAR_IRQ14	0 $\overline{\text{IRQ14}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 DDATA0

Address: 0xFC0A_405B (PAR_IRQ1DEBUGL) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	PAR_IRQ13	0	PAR_IRQ12	0	PAR_IRQ11	0	PAR_IRQ10
W								
Reset:	0	1	0	1	0	1	0	1

Figure 14-36. IRQ1 Pin Assignment Low (PAR_IRQ1DEBUGL)

Table 14-22. PAR_IRQ1DEBUGL Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 PAR_IRQ13	0 $\overline{\text{IRQ13}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 PST3
5	Reserved, must be cleared.
4 PAR_IRQ12	0 $\overline{\text{IRQ12}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 PST2
3	Reserved, must be cleared.
2 PAR_IRQ11	0 $\overline{\text{IRQ11}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 PST1
1	Reserved, must be cleared.
0 PAR_IRQ10	0 $\overline{\text{IRQ10}}$/GPIO. See Chapter 16, “Edge Port Modules (EPORTn),” for details. 1 PST0

14.3.5.8 SIM Port 1 Pin Assignment Registers (PAR_SIMP1H & PAR_SIMP1L)

The PAR_SIMP1H/L registers control the functions of the SIM port 1 pins.

Address: 0xFC0A_405C (PAR_SIMP1H) Access: User read/write

	7	6	5	4	3	2	1	0
R W	PAR_SIM_DATA1		PAR_SIM_VEN1		PAR_SIM_RST1		PAR_SIM_PD1	
Reset:	0	0	0	0	0	0	0	0

Figure 14-37. SIM Port 1 Pin Assignment High (PAR_SIMP1H)

Table 14-23. PAR_SIMP1H Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–6
PAR_DATA1
5–4
PAR_VEN1
3–2
PAR_RST1
1–0
PAR_PD1</td><td>SIM port 1 pin assignment. Configure the SIM port pins for one of their primary functions or GPIO.
<table>
<tr><th></th><th>PAR_DATA1</th><th>PAR_VEN1</th><th>PAR_RST1</th><th>PAR_PD1</th></tr>
<tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr>
<tr><td>01</td><td>U1TXD</td><td>U1RXD</td><td>$\overline{\text{U1RTS}}$</td><td>$\overline{\text{U1CTS}}$</td></tr>
<tr><td>10</td><td>SSI_TXD</td><td>SSI_RXD</td><td>SSI_FS</td><td>SSI_BCLK</td></tr>
<tr><td>11</td><td>SIM_DATA1</td><td>SIM_VEN1</td><td>SIM_RST1</td><td>SIM_PD1</td></tr>
</table></td></tr>
</table>

Address: 0xFC0A_405D (PAR_SIMP1L) Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_CLK1		0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-38. SIM Port 1 Pin Assignment Low (PAR_SIMP1L)

Table 14-24. PAR_SIMP1L Field Descriptions

Field	Description
7–6 PAR_CLK1	00 GPIO 01 Reserved 10 SSI_MCLK 11 SIM_CLK1
5–0	Reserved, must be cleared.

14.3.5.9 SIM Port 0 Pin Assignment Register (PAR_SIMP0)

The PAR_SIMP0 register controls the functions of the SIM port 0 pins.

Address: 0xFC0A_405E (PAR_SIMP0) Access: User read/write

	7	6	5	4	3	2	1	0
R				PAR_ SIM_DATA0	PAR_ SIM_VEN0	PAR_ SIM_RST0	PAR_ SIM_PD0	PAR_ SIM_CLK0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-39. SIM Port 0 Pin Assignment (PAR_SIMP0)

Table 14-25. PAR_SIMP0H Field Descriptions

Field	Description
7–5	Reserved, must be cleared.
4 PAR_DATA0 3 PAR_VEN0 2 PAR_RST0 1 PAR_PD0 0 PAR_CLK0	SIM port 1 pin assignment. Configure the SIM port pins for one of their primary functions or GPIO.

	PAR_DATA0	PAR_VEN0	PAR_RST0	PAR_PD0	PAR_CLK0
0	GPIO	GPIO	GPIO	GPIO	GPIO
1	SIM_DATA0	SIM_VEN0	SIM_RST0	SIM_PD0	SIM_CLK0

14.3.5.10 Timer Pin Assignment Registers (PAR_TIMER)

Address: 0xFC0A_405F (PAR_TIMER) Access: User read/write

	7	6	5	4	3	2	1	0
R	PAR_T3IN		PAR_T2IN		PAR_T1IN		PAR_T0IN	
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-40. Timer Pin Assignment (PAR_TIMER)

Table 14-26. PAR_TIMER Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–6
PAR_T3IN
5–4
PAR_T2IN
3–2
PAR_T1IN
1–0
PAR_T0IN</td><td>Timers pin assignment. Configures the timer pins for one of their primary functions or GPIO.
<table>
<tr><th></th><th>PAR_T3IN</th><th>PAR_T2IN</th><th>PAR_T1IN</th><th>PAR_T0IN</th></tr>
<tr><td>00</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr>
<tr><td>01</td><td>IRQ03</td><td>IRQ02</td><td>$\overline{\mathrm{DACK1}}$</td><td>CODEC_ALTCLK</td></tr>
<tr><td>10</td><td>T3OUT</td><td>T2OUT</td><td>T1OUT</td><td>T0OUT</td></tr>
<tr><td>11</td><td>T3IN</td><td>T2IN</td><td>T1IN</td><td>T0IN</td></tr>
</table>
</td></tr>
</table>

14.3.5.11 UART Pin Assignment Register (PAR_UART)

The PAR_UART register controls the functions of the UART pins.

Address: 0xFC0A_4060 (PAR_UART) Access: User read/write

<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R
W</td><td>PAR_
U2TXD</td><td>PAR_
U2RXD</td><td>PAR_
U0TXD</td><td>PAR_
U0RXD</td><td colspan="2">PAR_
U0RTS</td><td colspan="2">PAR_
U0CTS</td></tr>
<tr><td>Reset:</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 14-41. UART Pin Assignment (PAR_UART)

Table 14-27. PAR_UART Field Descriptions

Field	Description
7 PAR_U2TXD	0 GPIO 1 U2TXD
6 PAR_U2RXD	0 GPIO 1 U2RXD
5 PAR_U0TXD	0 GPIO 1 U0TXD
4 PAR_U0RXD	0 GPIO 1 U0RXD
3–2 PAR_U0RTS	00 GPIO 01 Reserved 10 USBO_VBUS_OC 11 $\overline{\mathrm{U0RTS}}$
1–0 PAR_U0CTS	00 GPIO 01 USB_PULLUP 10 USBO_VBUS_EN 11 $\overline{\mathrm{U0CTS}}$

14.3.5.12 Debug Pin Assignment Registers (PAR_DEBUG)

The PAR_DEBUG register controls the functions of the ALLPST pin.

Address: 0xFC0A_4062 (PAR_DEBUG) Access: User read/write

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

Figure 14-42. Debug Pin Assignment (PAR_DEBUG)

Table 14-28. PAR_DEBUG Field Descriptions

Field	Description
7 PAR_ALLPST	0 GPIO 1 ALLPST
6–0	Reserved, must be cleared.

14.3.5.13 eSDHC Pin Assignment Registers (PAR_SDHC)

The PAR_SDHC register controls the functions of the eSDHC pins.

Address: 0xFC0A_4063 (PAR_SDHC) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	PAR_ SDHC_DATA3	PAR_ SDHC_DATA2	PAR_ SDHC_DATA1	PAR_ SDHC_DATA0	PAR_ SDHC_CMD	PAR_ SDHC_CLK
W								
Reset:	0	0	0	0	0	0	0	0

Figure 14-43. SDHC Pin Assignment (PAR_SDHC)

Table 14-29. PAR_SDHC Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7–6</td><td>Reserved, must be cleared.</td></tr>
<tr><td>5–2
PAR_DATA[3:0]
1
PAR_CMD
0
PAR_CLK</td><td>eSDHC pin assignment. Configures the eSDHC pins for their primary function or GPIO.
<table>
<tr><th></th><th>PAR_DATA3</th><th>PAR_DATA2</th><th>PAR_DATA1</th><th>PAR_DATA0</th><th>PAR_CLK</th></tr>
<tr><td>0</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td><td>GPIO</td></tr>
<tr><td>1</td><td>SDHC_DATA3</td><td>SDHC_DATA2</td><td>SDHC_DATA1</td><td>SDHC_DATA0</td><td>SDHC_CLK</td></tr>
</table>
</td></tr>
</table>

14.3.5.14 SSI Pin Assignment Registers (PAR_SSIH & PAR_SSIL)

The PAR_SSIH/L registers control the functions of the SSI pins.

Address: 0xFC0A_4064 (PAR_SSIH) Access: User read/write

	7	6	5	4	3	2	1	0
R W	PAR_RXD		PAR_TXD		PAR_FS		PAR_MCLK	
Reset:	0	0	0	0	0	0	0	0

Figure 14-44. SSI Pin Assignment High (PAR_SSIH)

Table 14-30. PAR_SSIH Field Descriptions

Field	Description
7–6 PAR_RXD 5–4 PAR_TXD 3–2 PAR_FS 1–0 PAR_MCLK	SSI pin assignment. Configures the SSI pins for one of their primary functions or GPIO.

	PAR_RXD	PAR_TXD	PAR_FS	PAR_MCLK
00	GPIO	GPIO	GPIO	GPIO
01	U1RXD	U1TXD	$\overline{\text{U1RTS}}$	SSI_CLKIN
10	Reserved	Reserved	Reserved	Reserved
11	SSI_RXD	SSI_TXD	SSI_FS	SSI_MCLK

Address: 0xFC0A_4065 (PAR_SSIL) Access: User read/write

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

Figure 14-45. SSI Pin Assignment Low (PAR_SSIL)

Table 14-31. PAR_SSIL Field Descriptions

Field	Description
7–6 PAR_BCLK	SSI_BCLK pin assignment. Configures the SSI_BCLK pin for one of its primary functions or GPIO. 00 GPIO 01 $\overline{\text{U1CTS}}$ 10 Reserved 11 SSI_BCLK
5–0	Reserved, must be cleared.

14.3.6 Mode Select Control Registers (MSCR*n*)

The MSCR*n* registers select the mode of the following groups of FlexBus and SDRAM pins:

Table 14-32. Mode Select Control Registers

Register	Description	Affected Pins
MSCR1	Shared address and byte strobes	FB_A[15:11, 9:0], and FB_BE/BWE[3:0]
MSCR2	SDRAM pins	SD_A10, SD_CAS, SD_CKE, SD_CLK, SD_CLK, SD_CS0, SD_DQS[3:0], SD_RAS, and SD_WE
MSCR3	FlexBus address and control pins	FB_A[23:16, 10], FB_OE, FB_R/W, FB_TA (never driven by the FlexBus but available as GPIO), FB_TS, FB_CS[5:4,1:0], FB_CLK
MSCR4	Upper data bus pins	FB_D[31:16] / SD_D[31:16]
MSCR5	Lower data bus pins	FB_D[15:0] / FB_D[31:16]/SD_D[15:0]

Address: 0xFC0A_4068 (MSCR1)
0xFC0A_4069 (MSCR2)
0xFC0A_406A (MSCR3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	MSCR*n*			0	0	0	0	0
W								
Reset:	1	1	0	0	0	0	0	0

Figure 14-46. Mode Select Control Register *n* (MSCR1–3)

Table 14-33. MSCR1–3 Field Descriptions

Field	Description
7–5 MSCR*n*	Drive strength mode. See Table 14-32 for the pins affected by each MSCR*n* register. 000 Half strength 1.8V mobile DDR 001 Full strength 1.8V mobile DDR 011 2.5V DDR1 111 3.3V SDR Else Reserved
4–0	Reserved, must be cleared.

Address: 0xFC0A_406B (MSCR45)

Access: User read/write

	7	6	5	4	3	2	1	0
R	MSCR4			MSCR5			0	0
W								
Reset:	1	1	0	1	1	0	0	0

Figure 14-47. Mode Select Control Register 4 & 5 (MSCR45)

Table 14-34. MSCR45 Field Descriptions

Field	Description
7–5 MSCR4	Drive strength mode for the upper data bus pins. See Table 14-32 for the exact pins affected 000 Half strength 1.8V mobile DDR 001 Full strength 1.8V mobile DDR 011 2.5V DDR1 111 3.3V SDR Else Reserved
4–2 MSCR5	Drive strength mode for the lower data bus pins. See Table 14-32 for the exact pins affected 000 Half strength 1.8V mobile DDR 001 Full strength 1.8V mobile DDR 011 2.5V DDR1 111 3.3V SDR Else Reserved
1–0	Reserved, must be cleared.

14.3.7 Slew Rate Control Registers (SRCR_*x*)

SRCR_x controls the slew rate of some pins on the device.

Address: 0xFC0A_406C (SRCR_DSPI)
0xFC0A_406E (SRCR_I2C)
0xFC0A_4074 (SRCR_SDHC)
0xFC0A_4075 (SRCR_SSI)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	SRE_*x*	
W								
Reset:	0	0	0	0	0	0	See Note	0

Note: Reset state is the value of CCR[LOAD].

Figure 14-48. Port *x* Slew Rate Control Register (SRCR_*x*)

Address: 0xFC0A_406F (SRCR_IRQ)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	SRE_IRQ0		SRE_IRQ1DEBUG	
W								
Reset:	0	0	0	0	See Note	0	See Note	See Note

Note: SRE_IRQ1DEBUG does not affect the IRQ1FEC*n* signals
Reset state is the value of CCR[LOAD].

Figure 14-49. Edge Port Slew Rate Control Register (SRCR_IRQ)

Address: 0xFC0A_4070 (SRCR_SIM) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	SRE_SIMP0		SRE_SIMP1	
W								
Reset:	0	0	0	0	See Note	0	See Note	0

Note: Reset state is the value of CCR[LOAD].

Figure 14-50. SIM Slew Rate Control Register (SRCR_SIM)

Address: 0xFC0A_4071 (SRCR_TIMER) Access: User read/write

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

Note: Reset state is the value of CCR[LOAD].

Figure 14-51. Timer Slew Rate Control Register (SRCR_TIMER)

Address: 0xFC0A_4072 (SRCR_UART) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	SRE_UART2		SRE_UART0	
W								
Reset:	0	0	0	0	0	See Note	0	See Note

Note: Reset state is the value of CCR[LOAD].

Figure 14-52. UART Slew Rate Control Register (SRCR_UART)

Table 14-35. SRCR_*x* Field Descriptions

Field	Description
SRE_*x*	Slew rate control. 00 Lowest slew rate 01 Low slew rate 10 High slew rate 11 Highest slew rate

Note: See above figures for bit field positions.

Table 14-36. SRCR_*x* Pins Affected

SRCR_*x*	Signals Affected
SRE_DSPI	DSPI_PCS3, DSPI_PCS2, DSPI_PCS1, DSPI_PCS0, DSPI_SIN, DSPI_SOUT, DSPI_SCK
SRE_I2C	I2C_SDA, I2C_SCL

Table 14-36. SRCR_*x* Pins Affected (continued)

SRCR_*x*	Signals Affected
SRE_IRQ1	IRQ1[7:0]
SRE_IRQ0	IRQ0[7,6,4,1]
SRE_SIM1	SIM_PD1, SIM_RST1, SIM_CLK1, SIM_VEN1, SIM_DATA1
SRE_SIM0	SIM_CLK0, SIM_PD0, SIM_RST0, SIM_VEN0. SIM_DATA0
SRE_TIMER	T3IN, T2IN, T1IN, T0IN
SRE_UART2	U2TXD, U2RXD
SRE_UART0	$\overline{\text{U0\_CTS}}$, $\overline{\text{U0\_RTS}}$, U0_RXD, U0_TXD
SRE_SDHC	SDHC_CLK, SDHC_CMD, SDHC_DAT[3:0]
SRE_SSI	SSI_BCLK, SSI_FS, SSI_MCLK, SSI_RXD, SSI_TXD

14.3.8 Drive Strength Control Registers (DSCR_*x*)

DSCR_FEC controls the drive strength to the RMII pins.

Address: 0xFC0A_406D (DSCR_FEC) Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	DSE_RMII_CLK		DSE_RMII0		DSE_RMII1	
W								
Reset:	0	0	0	See Note	0	See Note	0	See Note

Note: Reset state is the value of CCR[LOAD].

Figure 14-53. FEC Drive Strength Control Register (DSCR_FEC)

Table 14-37. DSCR_*x* Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–4 DSE_RMII_CLK 3–2 DSE_RMII0 1–0 DSE_RMII1	Drive strength control. Controls the drive strength of the various pins. See Table 14-38 for a list of the pins affected for each register bit field. 00 10pF 01 20pF 10 30pF 11 50pF

Table 14-38. DSCR_*x* Pins Affected

DSCR_FEC	Signals Affected
DSE_RMII_CLK	RMII_REF_CLK
DSE_RMII0	RMII0_CRS_DV, RMII0_RXD[1:0], RMII0_RXER, RMII0_TXD[1:0], RMII0_TXEN, RMII0_MDC, RMII0_MDIO
DSE_RMII1	RMII1_CRS_DV, RMII1_RXD[1:0], RMII1_RXER, RMII1_TXD[1:0], RMII1_TXEN, RMII1_MDC, RMII1_MDIO

14.3.9 Pull Control Registers (PCRH & PCRL)

A few pins on the device have configurable pull-ups/downs. The pull control registers select their direction and enable them.

The rest of the pins with pull-ups/downs have their pulls enabled only when the primary functions are enabled on those pins. There are no control bits associated with the pulls on these pins.

Address: 0xFC0A_4078 (PCRH) Access: User read/write

	7	6	5	4	3	2	1	0
R	EN_ DSPI_PCS0	EN_ SIM_VEN1	DIR_ SIM_VEN1	EN_ SIM_DATA1	DIR_ SIM_DATA1	0	EN_ SSI	DIR_ SSI
W								
Reset:	1	1	1	1	0	0	1	1

Figure 14-54. Pull Control Register High (PCRH)

Table 14-39. PCRH Field Descriptions

Field	Description
7 EN_DSPI_PCS0	Enables the pull-up for the pin DSPI_PCS0/$\overline{\text{SS}}$. This bit is effective only when the pin is functioning as DSPI_$\overline{\text{SS}}$. For all other functions, the pull-up is disabled 0 Pull-up disabled 1 Pull-up enabled
6 EN_SIM_VEN1	Enables the pull-up/down for the SIM_VEN1 pin’s SSI_RXD function. This bit is effective only when this pin is functioning as SSI_RXD. For all other functions, the pull-up/down is disabled. 0 Pull-up/down disabled 1 Pull-up/down enabled
5 DIR_SIM_VEN1	Selects the direction of the pull on SIM_VEN1 when configured as SSI_RXD. This bit is ignored if EN_SIM_VEN1 is cleared. 0 Pull-down 1 Pull-up
4 EN_SIM_DATA1	Enables the pull-up/down for the SIM_DATA1 pin’s SSI_TXD function. This bit is effective only when this pin is functioning as SSI_TXD. For all other functions on this pin, the pull-up/down is disabled. 0 Pull-up/down disabled 1 Pull-up/down enabled

Table 14-39. PCRH Field Descriptions (continued)

Field	Description
3 DIR_SIM_DATA1	Selects the direction of the pull on SIM_DATA1 when configured as SSI_TXD. This bit is ignored if EN_SIMDATA1 is cleared. 0 Pull-down 1 Pull-up
2	Reserved, must be cleared.
1 EN_SSI	Enables the pull-up/down for SSI data pins when functioning as SSI_RXD and SSI_TXD. For all other functions on these pins the pull-up/down's are disabled.
0 DIR_SSI	Selects the direction of the pull on the SSI data pins when configured as SSI_RXD and SSI_TXD. This bit is ignored if EN_SSI is cleared. 0 Pull-down 1 Pull-up

Address: 0xFC0A_4079 (PCRL) Access: User read/write

	7	6	5	4	3	2	1	0
R	EN_ SDHC_DAT3	DIR_ SDHC_DAT3	EN_ SDHC_DAT2	EN_ SDHC_DAT1	EN_ SDHC_DAT0	EN_ SDHC_CMD	0	0
W								
Reset:	1	1	1	1	1	1	0	0

Figure 14-55. Pull Control Register Low (PCRL)

Table 14-40. PCRL Field Descriptions

Field	Description
7 EN_SDHC_DAT3	Enables the pull-up/down for the SDHC_DAT3 pin, when functioning as SDHC_DAT3. For all other functions on this pin, the pull-up/down is disabled. 0 Pull-up disabled 1 Pull-up enabled
6 DIR_SDHC_DAT3	Selects the direction of the pull on SDHC_DAT3 when configured as SDHC_DAT3. This bit is ignored if EN_SDHC_DAT3 is cleared. 0 Pull-down 1 Pull-up
5 EN_SDHC_DAT2	Enables the pull-up for the SDHC_DAT2 pin, when functioning as SDHC_DAT2. For all other functions on this pin, the pull-up is disabled. 0 Pull-up disabled 1 Pull-up enabled
4 EN_SDHC_DAT1	Enables the pull-up for the SDHC_DAT1 pin, when functioning as SDHC_DAT1. For all other functions on this pin, the pull-up is disabled. 0 Pull-up disabled 1 Pull-up enabled
3 EN_SDHC_DAT0	Enables the pull-up for the SDHC_DAT0 pin, when functioning as SDHC_DAT0. For all other functions on this pin, the pull-up is disabled. 0 Pull-up disabled 1 Pull-up enabled

Table 14-40. PCRL Field Descriptions (continued)

Field	Description
2 EN_SDHC_CMD	Enables the pull-up for the SDHC_CMD pin, when functioning as SDHC_CMD. For all other functions on this pin, the pull-up is disabled. 0 Pull-up disabled 1 Pull-up enabled
1–0	Reserved, must be cleared.

14.4 Functional Description

14.4.1 Overview

Initial pin function is determined during reset configuration. The pin assignment registers 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, allowing access to external boot memory.

Every GPIO pin is individually configurable as an input or 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 GPIO, 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.

14.4.2 Port Digital I/O Timing

Input data on all pins configured as general purpose input is synchronized to the rising edge of FB_CLK, as shown in Figure 14-56.

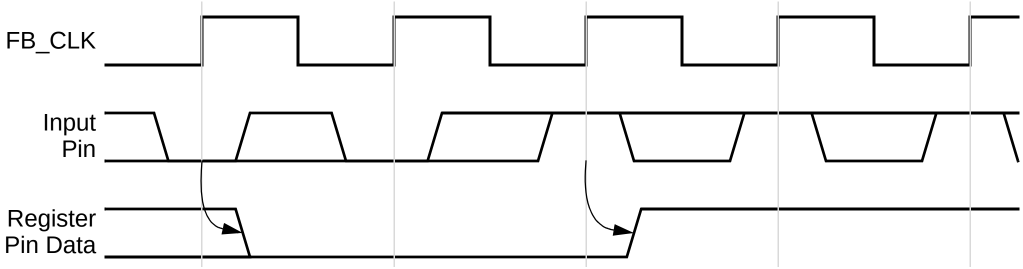


Figure 14-56. General Purpose Input Timing

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 14-57.

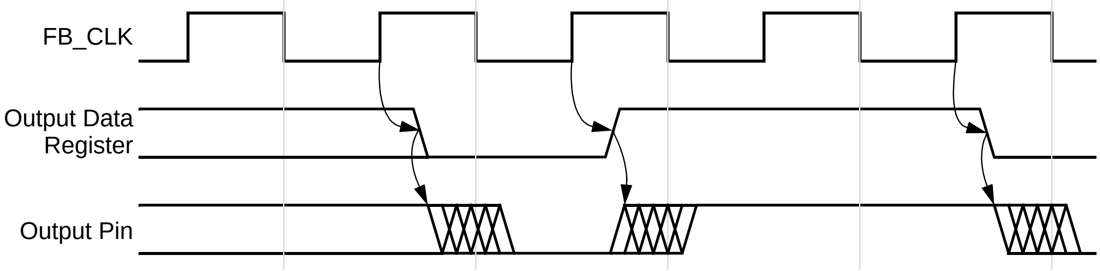


Figure 14-57. General Purpose Output Timing

14.5 Initialization/Application Information

The initialization for the pin multiplexing and control is done during reset configuration. All registers are reset to a predetermined state. Refer to Section 14.3, “Memory Map/Register Definition,” for more details on reset and initialization.

Chapter 15
Interrupt Controller Modules

15.1 Introduction

This section details the functionality of the interrupt controllers (INTC0, INTC1). The general features of the interrupt controller block include:

- 128 fully-programmable interrupt sources. Not all possible interrupt source locations are used on this device
- Each of the sources has a unique interrupt control register (ICR0*n*, ICR1*n*) to define the software-assigned levels
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source, plus global mask-all capability
- Supports hardware and software interrupt acknowledge cycles
- Wake-up signal from low-power stop modes

The 64, fully-programmable interrupt sources for the two interrupt controllers manage the complete set of interrupt sources from all of the modules on the device. This section describes how the interrupt sources are mapped to the interrupt controller logic and how interrupts are serviced.

15.1.1 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 15.3.1.3, "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.

15.2 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 15-2. The base addresses for the interrupt controllers are listed below.

Table 15-1. 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 15.2.10, "Software and Level 1–7 IACK Registers (SWIACKn, L1IACKn–L7IACKn)" for more information

Table 15-2. 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	15.2.1/15-4
0xFC04_8004	Interrupt Pending Register Low (IPRL0)	32	R	0x0000_0000	15.2.1/15-4
0xFC04_8008	Interrupt Mask Register High (IMRH0)	32	R/W	0xFFFF_FFFF	15.2.2/15-5
0xFC04_800C	Interrupt Mask Register Low (IMRL0)	32	R/W	0xFFFF_FFFF	15.2.2/15-5
0xFC04_8010	Interrupt Force Register High (INTFRCH0)	32	R/W	0x0000_0000	15.2.3/15-6
0xFC04_8014	Interrupt Force Register Low (INTFRCL0)	32	R/W	0x0000_0000	15.2.3/15-6
0xFC04_801A	Interrupt Configuration Register (ICONFIG)	16	R/W	0x0000	15.2.4/15-7
0xFC04_801C	Set Interrupt Mask (SIMR0)	8	W	0x00	15.2.5/15-8
0xFC04_801D	Clear Interrupt Mask (CIMR0)	8	W	0x00	15.2.6/15-9
0xFC04_801E	Current Level Mask (CLMASK)	8	R/W	0x0F	15.2.7/15-9
0xFC04_801F	Saved Level Mask (SLMASK)	8	R/W	0x0F	15.2.8/15-10
0xFC04_8040 + *n* (*n*=0:63)	Interrupt Control Registers (ICR0*n*)	8	R/W	0x00	15.2.9/15-11
0xFC04_80E0	Software Interrupt Acknowledge (SWIACK0)	8	R	0x00	15.2.10/15-15
0xFC04_80E0 + 4*n* (*n*=1:7)	Level *n* Interrupt Acknowledge Registers (L*n*IACK0)	8	R	0x18	15.2.10/15-15
Interrupt Controller 1					
0xFC04_C000	Interrupt Pending Register High (IPRH1)	32	R	0x0000_0000	15.2.1/15-4
0xFC04_C004	Interrupt Pending Register Low (IPRL1)	32	R	0x0000_0000	15.2.1/15-4
0xFC04_C008	Interrupt Mask Register High (IMRH1)	32	R/W	0xFFFF_FFFF	15.2.2/15-5
0xFC04_C00C	Interrupt Mask Register Low (IMRL1)	32	R/W	0xFFFF_FFFF	15.2.2/15-5
0xFC04_C010	Interrupt Force Register High (INTFRCH1)	32	R/W	0x0000_0000	15.2.3/15-6
0xFC04_C014	Interrupt Force Register Low (INTFRCL1)	32	R/W	0x0000_0000	15.2.3/15-6
0xFC04_C01C	Set Interrupt Mask (SIMR1)	8	W	0x00	15.2.5/15-8
0xFC04_C01D	Clear Interrupt Mask (CIMR1)	8	W	0x00	15.2.5/15-8
0xFC04_C040 + *n* (*n*=1:63)	Interrupt Control Registers (ICR1*n*)	8	R/W	0x00	15.2.9/15-11
0xFC04_C0E0	Software Interrupt Acknowledge (SWIACK1)	8	R	0x00	15.2.10/15-15
0xFC04_C0E0 + 4*n* (*n*=1:7)	Level *n* Interrupt Acknowledge Registers (L*n*IACK1)	8	R	0x18	15.2.10/15-15
Global IACK Registers					

Table 15-2. Interrupt Controller Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC05_40E0	Global Software Interrupt Acknowledge (GSWIACK)	8	R	0x00	15.2.10/15-15
0xFC05_40E0 + 4*n* (*n*=1:7)	Global Level *n* Interrupt Acknowledge Registers (GL*n*IACK)	8	R	0x18	15.2.10/15-15

15.2.1 Interrupt Pending Registers (IPRH*n*, IPRL*n*)

The IPRH*n* and IPRL*n* registers, Figure 15-1 and Figure 15-2, are each 32 bits in size, and provide a bit map for each interrupt request to indicate if there is an active request (1 equals active request, 0 equals no request) for the given source. The interrupt mask register state does not affect the IPR*n*. The IPR*n* is cleared by reset and is a read-only register, so any attempted write to this register is ignored.

Address 0xFC04_8000 (IPRH0)
0xFC04_C000 (IPRH1)

Access: User read-only

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

Figure 15-1. Interrupt Pending Register High (IPRH*n*)

Table 15-3. IPRH*n* Field Descriptions

Field	Description
31–0 INT	Interrupt pending. Each bit corresponds to an interrupt source. The corresponding IMRH*n* bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRH*n* samples the signal generated by the interrupting source. The corresponding IPRH*n* bit reflects the state of the interrupt signal even if the corresponding IMRH*n* bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

Address 0xFC04_8004 (IPRL0)
0xFC04_C004 (IPRL1)

Access: User read-only

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

Figure 15-2. Interrupt Pending Register Low (IPRL*n*)

Table 15-4. 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

15.2.2 Interrupt Mask Register (IMRH*n*, IMRL*n*)

The IMRH*n* and IMRL*n* registers are each 32 bits in size and provide a bit map for each interrupt to allow the request to be disabled (1 equals disable the request, 0 equals enable the request). The IMRL register is used for masking interrupt sources 0 to 31, while the IMRH register is used for masking interrupts 32 to 63. The IMR*n* is set to all ones by reset, disabling all interrupt requests. The IMR*n* can be read and written.

NOTE

A spurious interrupt may occur if an interrupt source is being masked in the interrupt controller mask register (IMR) or a module's interrupt mask register while the interrupt mask in the status register (SR[I]) is set to a value lower than the interrupt's level. This is because by the time the status register acknowledges this interrupt, the interrupt has been masked. A spurious interrupt is generated because the CPU cannot determine the interrupt source. To avoid this situation for interrupts sources with levels 1-6, first write a higher level interrupt mask to the status register, before setting the mask in the IMR or the module's interrupt mask register. After the mask is set, return the interrupt mask in the status register to its previous value. Because level 7 interrupts cannot be disabled in the status register prior to masking, use of the IMR or module interrupt mask registers to disable level 7 interrupts is not recommended.

Address 0xFC04_8008 (IMRH0)
0xFC04_C008 (IMRH1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	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 15-3. Interrupt Mask Register High (IMRH*n*)

Table 15-5. IMRH*n* Field Descriptions

Field	Description
31–0 INT_MASK	Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRH*n* bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRH*n* bit reflects the state of the interrupt signal even if the corresponding IMRH*n* bit is set. 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked

Address 0xFC04_800C (IMRL0)
0xFC04_C00C (IMRL1)

Access: User read/write

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

Figure 15-4. Interrupt Mask Register Low (IMRL*n*)

Table 15-6. 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

15.2.3 Interrupt Force Registers (INTFRCH*n*, INTFRCL*n*)

The INTFRCH*n* and INTFRCL*n* registers are each 32 bits in size and provide a mechanism to allow software generation of interrupts for each possible source for functional or debug purposes. The system design may reserve one or more sources to allow software to self-schedule interrupts by forcing one or more of these bits (set to force request, clear to negate request) in the appropriate INTFRC*n* register. The INTFRCL*n* register forces interrupts for sources 0 to 31, while the INTFRCH*n* register forces interrupts for sources 32 to 63. The assertion of an interrupt request via the interrupt force register is not affected by the interrupt mask register. The INTFRC*n* registers are cleared by reset.

Address 0xFC04_8010 (INTFRCH0)
0xFC04_C010 (INTFRCH1)

Access: User read/write

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

Figure 15-5. Interrupt Force Register High (INTFRCH*n*)

Table 15-7. INTFRCH*n* Field Descriptions

Field	Description
31–0 INTFRCH	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on the corresponding interrupt source 1 Force an interrupt on the corresponding source

Address 0xFC04_8014 (INTFRCL0)
0xFC04_C014 (INTFRCL1)

Access: User read/write

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

Figure 15-6. Interrupt Force Register Low (INTFRCL*n*)

Table 15-8. 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

15.2.4 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 15-7. Interrupt Configuration Register (ICONFIG)

Table 15-9. 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.

15.2.5 Set Interrupt Mask Register (SIMR*n*)

The SIMR*n* register provides a simple mechanism to set a given bit in the IMR*n* registers to mask the corresponding interrupt request. The value written to the SIMR field causes the corresponding bit in the IMR*n* register to be set. The SIMR*n*[SALL] bit provides a global set function, forcing the entire contents of IMR*n* to be set, thus masking all interrupts. Reads of this register return all zeroes. This register is provided so interrupt service routines can easily mask the given interrupt request without the need to perform a read-modify-write sequence on the IMR*n* register.

Address: 0xFC04_801C (SIMR0)
0xFC04_C01C (SIMR1)

Access: User write-only

	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 15-8. Set Interrupt Mask Register (SIMR*n*)

Table 15-10. 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.

15.2.6 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 15-9. Clear Interrupt Mask Register (CIMR*n*)

Table 15-11. 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.

15.2.7 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 15-10. Current Level Mask Register (CLMASK)

Table 15-12. 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.

15.2.8 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 15-11. Saved Level Mask Register (SLMASK)

Table 15-13. 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.

15.2.9 Interrupt Control Register (ICR0*n*, ICR1*n*, (*n* = 00, 01, 02, ..., 63))

Each ICR register specifies the interrupt level (1–7) for the corresponding interrupt source. These registers are cleared by reset and should be programmed with the appropriate levels before interrupts are enabled.

When multiple interrupt requests are programmed to the same level number, they are processed in a descending request number order. As an example, if requests 63, 62, 2, and 1 are programmed to a common level, request 63 is processed first, then request 62, then request 2, and finally request 1.

This definition allows software maximum flexibility in grouping interrupt request sources within any given priority level. The priority level in the ICRs directly corresponds to the interrupt level supported by the ColdFire processor.

Address: 0xFC04_8040 – 7F (ICR000 – ICR063)
0xFC04_C040 – 7F (ICR100 – ICR163)   Access: User read/write

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

Figure 15-12. Interrupt Control Registers (ICR0*n*, ICR1*n*)

Table 15-14. ICR*n* Field Descriptions

Field	Description
7–3	Reserved, must be cleared.
2–0 LEVEL	Interrupt level. Indicates the interrupt level assigned to each interrupt input. A level of 0 effectively disables the interrupt request, while a level 7 interrupt is given the highest priority. If interrupt masking is enabled (ICONFIG[EMASK] = 1), the acknowledgement of a level-*n* request forces the controller to automatically mask all interrupt requests of level-*n* and lower.

15.2.9.1 Interrupt Sources

Table 15-15 and Table 15-16 list the interrupt sources for each interrupt request line for INTC0 and INTC1.

Table 15-15. Interrupt Source Assignment For INTC0

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0	Not Used			
1	EPORT 0	EPFR0[EPF1]	Edge port 0 flag 1	Write EPF1 = 1
2		EPFR0[EPF2]	Edge port 0 flag 2	Write EPF2 = 1
3		EPFR0[EPF3]	Edge port 0 flag 3	Write EPF3 = 1
4		EPFR0[EPF4]	Edge port 0 flag 4	Write EPF4 = 1
5		Not Used		
6		EPFR0[EPF6]	Edge port 0 flag 6	Write EPF6 = 1
7		EPFR0[EPF7]	Edge port 0 flag 7	Write EPF7 = 1
8	DMA	EDMA_INTR[INT00]	DMA Channel 0 transfer complete	Write EDMA_CINTR[CINT] = 0
9		EDMA_INTR[INT01]	DMA Channel 1 transfer complete	Write EDMA_CINTR[CINT] = 1
10		EDMA_INTR[INT02]	DMA Channel 2 transfer complete	Write EDMA_CINTR[CINT] = 2
11		EDMA_INTR[INT03]	DMA Channel 3 transfer complete	Write EDMA_CINTR[CINT] = 3
12		EDMA_INTR[INT04]	DMA Channel 4transfer complete	Write EDMA_CINTR[CINT] = 4
13		EDMA_INTR[INT05]	DMA Channel 5 transfer complete	Write EDMA_CINTR[CINT] = 5
14		EDMA_INTR[INT06]	DMA Channel 6 transfer complete	Write EDMA_CINTR[CINT] = 6
15		EDMA_INTR[INT07]	DMA Channel 7 transfer complete	Write EDMA_CINTR[CINT] = 7
16		EDMA_INTR[INT08]	DMA Channel 8 transfer complete	Write EDMA_CINTR[CINT] = 8
17		EDMA_INTR[INT09]	DMA Channel 9 transfer complete	Write EDMA_CINTR[CINT] = 9
18		EDMA_INTR[INT10]	DMA Channel 10 transfer complete	Write EDMA_CINTR[CINT] = 10
19		EDMA_INTR[INT11]	DMA Channel 11 transfer complete	Write EDMA_CINTR[CINT] = 11
20		EDMA_INTR[INT12]	DMA Channel 12 transfer complete	Write EDMA_CINTR[CINT] = 12
21		EDMA_INTR[INT13]	DMA Channel 13 transfer complete	Write EDMA_CINTR[CINT] = 13
22		EDMA_INTR[INT14]	DMA Channel 14 transfer complete	Write EDMA_CINTR[CINT] = 14
23		EDMA_INTR[INT15]	DMA Channel 15 transfer complete	Write EDMA_CINTR[CINT] = 15
24		EDMA_ERR[ERR*n*]	DMA Error Interrupt	Write EDMA_CERR[CERR] = *n*
25	SCM	SCMIR[CWIC]	Core Watchdog Timeout	Write SCMISR[CWIC] = 1
26	UART0	UISR0 register	UART0 Interrupt Request	Automatically cleared
27	UART1	UISR1 register	UART1 Interrupt Request	Automatically cleared
28	UART2	UISR2 register	UART2 Interrupt Request	Automatically cleared
29	Not Used			

Table 15-15. Interrupt Source Assignment For INTC0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
30	I^{2}C	I2SR[IIF]	I^{2}C Interrupt	Write I2SR[IIF] = 0
31	DSPI	DSPI_SR register	DSPI interrupt (Logical OR of INTC1's source #33–39)	Write 1 to appropriate DSPI_SR bit
32	DTIM0	DTER0 register	Timer 0 interrupt	Write 1 to appropriate DTER0 bit
33	DTIM1	DTER1 register	Timer 1 interrupt	Write 1 to appropriate DTER1 bit
34	DTIM2	DTER2 register	Timer 2 interrupt	Write 1 to appropriate DTER2 bit
35	DTIM3	DTER3 register	Timer 3 interrupt	Write 1 to appropriate DTER3 bit
36	FEC0	EIR0[TXF]	Transmit frame interrupt	Write EIR0[TXF] = 1
37		EIR0[TXB]	Transmit buffer interrupt	Write EIR0[TXB] = 1
38		EIR0[UN]	Transmit FIFO underrun	Write EIR0[UN] = 1
39		EIR0[RL]	Collision retry limit	Write EIR0[RL] = 1
40		EIR0[RXF]	Receive frame interrupt	Write EIR0[RXF] = 1
41		EIR0[RXB]	Receive buffer interrupt	Write EIR0[RXB] = 1
42		EIR0[MII]	MII interrupt	Write EIR0[MII] = 1
43		EIR0[LC]	Late collision	Write EIR0[LC] = 1
44		EIR0[HBERR]	Heartbeat error	Write EIR0[HBERR] = 1
45		EIR0[GRA]	Graceful stop complete	Write EIR0[GRA] = 1
46		EIR0[EBERR]	Ethernet bus error	Write EIR0[EBERR] = 1
47		EIR0[BABT]	Babbling transmit error	Write EIR0[BABT] = 1
48		EIR0[BABR]	Babbling receive error	Write EIR0[BABR] = 1
49	FEC1	EIR1[TXF]	Transmit frame interrupt	Write EIR1[TXF] = 1
50		EIR1[TXB]	Transmit buffer interrupt	Write EIR1[TXB] = 1
51		EIR1[UN]	Transmit FIFO underrun	Write EIR1[UN] = 1
52		EIR1[RL]	Collision retry limit	Write EIR1[RL] = 1
53		EIR1[RXF]	Receive frame interrupt	Write EIR1[RXF] = 1
54		EIR1[RXB]	Receive buffer interrupt	Write EIR1[RXB] = 1
55		EIR1[MII]	MII interrupt	Write EIR1[MII] = 1
56		EIR1[LC]	Late collision	Write EIR1[LC] = 1
57		EIR1[HBERR]	Heartbeat error	Write EIR1[HBERR] = 1
58		EIR1[GRA]	Graceful stop complete	Write EIR1[GRA] = 1
59		EIR1[EBERR]	Ethernet bus error	Write EIR1[EBERR] = 1
60		EIR1[BABT]	Babbling transmit error	Write EIR1[BABT] = 1
61		EIR1[BABR]	Babbling receive error	Write EIR1[BABR] = 1

Table 15-15. Interrupt Source Assignment For INTC0 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
62	SCM	SCMIR[CFEI]	Core bus error interrupt	Write SCMIR[CFEI] = 1
63	Not Used			

Table 15-16. Interrupt Source Assignment for INTC1

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0–24	Not Used			
25	EPORT 1	EPFR1[EPF0]	Edge port 1 flag 0	Write EPF0 = 1
26		EPFR1[EPF1]	Edge port 1 flag 1	Write EPF1 = 1
27		EPFR1[EPF2]	Edge port 1 flag 2	Write EPF2 = 1
28		EPFR1[EPF3]	Edge port 1 flag 3	Write EPF3 = 1
29		EPFR1[EPF4]	Edge port 1 flag 4	Write EPF4 = 1
30		EPFR1[EPF5]	Edge port 1 flag 5	Write EPF5 = 1
31		EPFR1[EPF6]	Edge port 1 flag 6	Write EPF6 = 1
32		EPFR1[EPF7]	Edge port 1 flag 7	Write EPF7 = 1
33	DSPI	DSPI_SR[EOQF]	End of queue interrupt	Write 1 to EOQF.
34		DSPI_SR[TFFF]	Transmit FIFO fill interrupt	Write 1 to TFFF.
35		DSPI_SR[TCF]	Transfer complete interrupt	Write 1 to TCF.
36		DSPI_SR[TFUF]	Transmit FIFO underflow interrupt	Write 1 to TFUF.
37		DSPI_SR[RFDF]	Receive FIFO not empty interrupt	Write 1 to RFDF after reading the DSPI_POPR register.
38		DSPI_SR[RFOF]	Receive FIFO overflow interrupt	Write 1 to RFOF.
39		DSPI_SR[RFOF] or DSPI_SR[TFUF]	Receive FIFO overflow or transmit FIFO underflow interrupt	Write 1 to either RFOF or TFUF.
40	RNG	EI	RNG interrupt flag	Write RNGCR[CI] = 1
41	PLL	PLL_SR[LOCF]	Loss of clock interrupt	Write 1 to LOCF
42		PLL_SR[LOLF]	Loss of lock interrupt	Write 1 to LOLF
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	IIM	IIM_SR or IIM_ESR	IIM interrupt	Write 1 to corresponding bit in IIM_SIMR or IIM_EIMR.

Table 15-16. Interrupt Source Assignment for INTC1 (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
51	Not Used			
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	Codec	CODEC_SR	Voice codec OR'd interrupt	Various, see below three codec interrupts.
55		CODEC_SR [VCRF or VCTE]	Voice codec interrupt	Reading CODEC_RX or writing CODEC_TX
56		CODEC_SR[VROE]	Voice codec receive overrun error interrupt	Reading CODEC_SR followed by reading CODEC_RX
57		CODEC_SR[VTUE]	Voice codec transmit underrun error interrupt	Reading CODEC_SR followed by reading CODEC_TX
58	Not Used			
59	SIM1	SIM_RSR or SIM_TSR	SIM data interrupt	Various, see chapter for details
60			SIM general interrupt	Various, see chapter for details
61–62	Not Used			
63	SDHC	IRQSTAT	SDHC OR'd interrupt	Write 1 to corresponding bit in IRQSTAT

15.2.10 Software and Level 1–7 IACK Registers (SWIACK*n*, L1IACK*n*–L7IACK*n*)

The eight IACK registers (per interrupt controller) can be explicitly addressed via the CPU, or implicitly addressed via a processor-generated interrupt acknowledge cycle during exception processing. In either case, the interrupt controller's actions are very similar.

First, consider an IACK cycle to a specific level: a level-*n* IACK. When this type of IACK arrives in the interrupt controller, the controller examines all the currently-active level *n* interrupt requests, determines the highest priority within the level, and then responds with the unique vector number corresponding to that specific interrupt source. The vector number is supplied as the data for the byte-sized IACK read cycle. In addition to providing the vector number, the interrupt controller also loads the level into the CLMASK register, where it may be retrieved later.

This interrupt controller design also supports the concept of a software IACK. A software IACK allows an interrupt service routine to determine if there are other pending interrupts so that the overhead associated with interrupt exception processing (including machine state save/restore functions) can be minimized. In general, the software IACK is performed near the end of an interrupt service routine, and if there are additional active interrupt sources, the current interrupt service routine (ISR) passes control to the appropriate service routine, but without taking another interrupt exception.

When the interrupt controller receives a software IACK read, it returns the vector number associated with the highest unmasked interrupt source for that interrupt controller. If there are no active sources, the interrupt controller returns an all-zero vector as the operand for the SWIACK register. A read from the

L*n*IACK registers when there are no active requests returns a value of 24 (0x18), signaling a spurious interrupt.

In addition to the software IACK registers in each interrupt controller, there are global software IACK registers. A read from the global SWIACK (GSWIACK) returns the vector number for the highest level and priority unmasked interrupt source from all interrupt controllers. A read from one of the global L*n*IACK (GL*n*IACK) registers returns the vector for the highest priority unmasked interrupt within a level for all interrupt controllers.

Address: 0xFC04_80E0 (SWIACK0)
0xFC04_80E0+4*n* (L*n*IACK0) *n*=1:7
0xFC04_C0E0 (SWIACK1)
0xFC04_C0E0+4*n* (L*n*IACK1) *n*=1:7
0xFC05_40E0 (GSWIACK)
0xFC05_40E0+4*n* (GL*n*IACK) *n*=1:7

Access: User read-only

	7	6	5	4	3	2	1	0
R	VECTOR							
W								
Reset (SWIACK*n*):	0	0	0	0	0	0	0	0
Reset (L*n*IACK*n*):	0	0	0	1	1	0	0	0

Figure 15-13. Software and Level *n* IACK Registers (SWIACK*n*, L1IACK*n* – L7IACK*n*)

Table 15-17. 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.

15.3 Functional Description

15.3.1 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 15.2.9.1, “Interrupt Sources”). The higher numbered interrupt source has priority over the lower numbered interrupt source. See the below table for an example.

Table 15-18. 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

15.3.1.1 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.

15.3.1.2 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.

15.3.1.3 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.

15.3.2 Prioritization Between Interrupt Controllers

The interrupt controllers have a fixed priority, where INTC0 has the highest priority, and INTC1 has the lowest priority. If both interrupt controllers have active interrupts at the same level, then the INTC0 interrupt is serviced first. If INTC1 has an active interrupt with a higher level than the highest INTC0 interrupt, then the INTC1 interrupt is serviced first.

15.3.3 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 ," 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 ”.

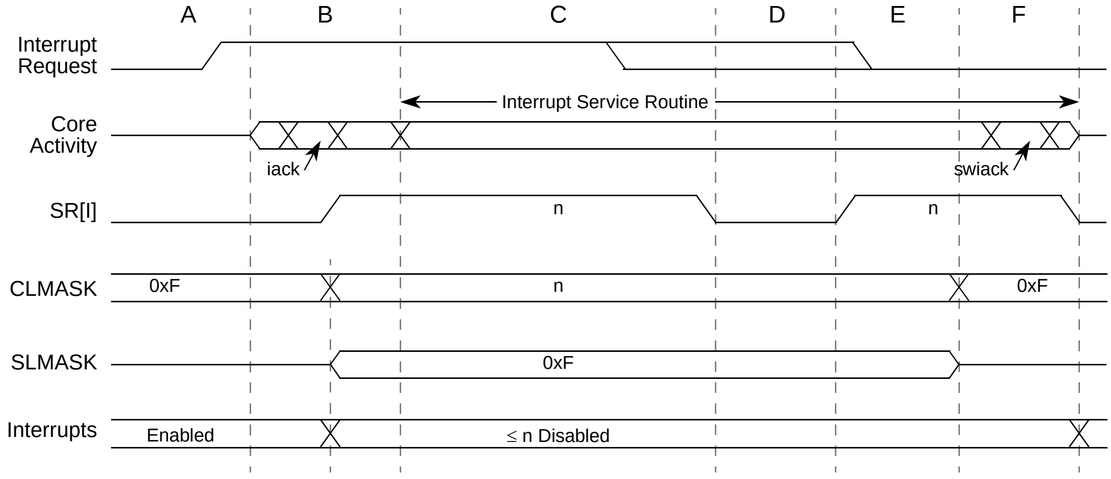
15.4 Initialization/Application Information

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.

15.4.1 Interrupt Service Routines

This section focuses on the interaction of the interrupt masking functionality with the service routine. Figure 15-14 presents a timing diagram showing various phases during the execution of an interrupt service routine with the controller level masking functionality enabled. The time scale in this diagram is not meant to be accurate.



Note: Not to scale

Figure 15-14. 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 16
Edge Port Modules (EPORT*n*)

16.1 Introduction

Although this device has two edge port modules, the description included herein treats each module as a single entity. Pay particular attention to the note below, as the two modules are not completely identical. Specifically, edge port module 0 has six interrupt inputs while module 1 contains eight.

The edge port module (EPORT) has up to eight interrupt pins, $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ0}}$. Each pin can be configured individually as a level-sensitive interrupt pin, an edge-detecting interrupt pin (rising edge, falling edge, or both), or a general-purpose input/output (I/O) pin.

NOTE

Not all EPORT signals may be output from the device. See Chapter 2, “Signal Descriptions,” to determine which signals are available.

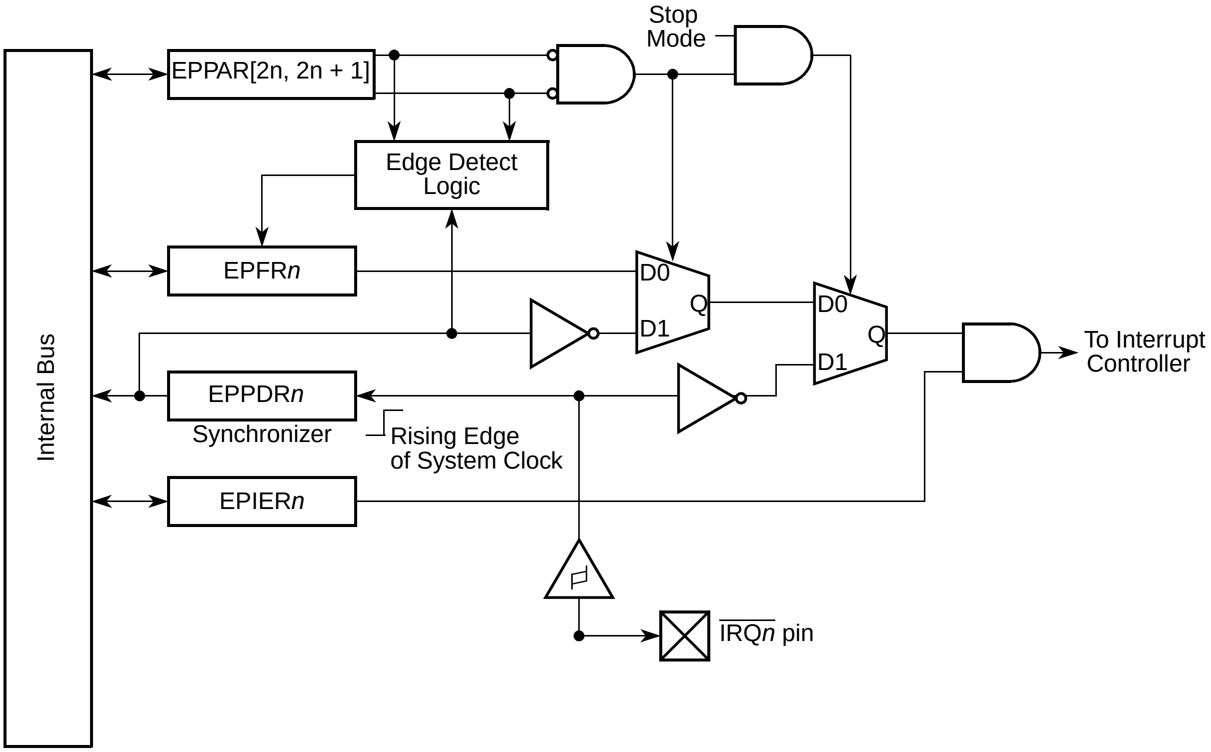


Figure 16-1. EPORT Block Diagram

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, “Pin-Multiplexing and Control”) prior to configuring the edge-port module.

16.2 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 16-1 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 16-1. Edge Port Module Operation in Low-Power Modes

Low-power Mode	EPORT Operation	Mode Exit
Wait	Normal	Any $\overline{\mathrm{IRQ}n}$ interrupt at or above level in WCR
Doze	Normal	Any $\overline{\mathrm{IRQ}n}$ interrupt at or above level in WCR
Stop	Level-sensing only	Any $\overline{\mathrm{IRQ}n}$ 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.

16.3 Signal Descriptions

All EPORT pins default to general-purpose input pins at reset. The pin value is synchronized to the rising edge of FB_CLK when read from the EPORT pin data register (EPPDR). The values used in the edge/level detect logic are also synchronized to the rising edge of FB_CLK. These pins use Schmitt-triggered input buffers with built-in hysteresis designed to decrease the probability of generating false, edge-triggered interrupts for slow rising and falling input signals.

When a pin is configured as an output, it is driven to a state whose level is determined by the corresponding bit in the EPORT data register (EPDR). All bits in the EPDR are set at reset.

16.4 Memory Map/Register Definition

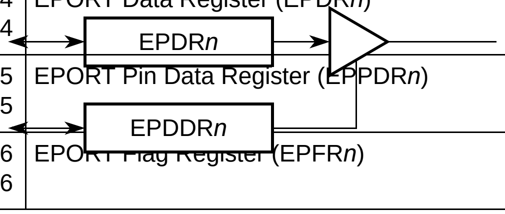
This subsection describes the memory map and register structure. Refer to Table 16-2 for a description of the EPORT memory map.

NOTE

> Longword accesses to any of the edge-port registers result in a bus error. Only byte and word accesses are allowed.

Table 16-2. Edge Port Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
EPORT0 **EPORT1**					
Supervisor Access Only Registers[1]					
0xFC09_0000 0xFC09_4000	EPORT Pin Assignment Register (EPPAR*n*)	16	R/W	0x0000	16.4.1/16-3
0xFC09_0002 0xFC09_4002	EPORT Data Direction Register (EPDDR*n*)	8	R/W	0x00	16.4.2/16-4
0xFC09_0003 0xFC09_4003	EPORT Interrupt Enable Register (EPIER*n*)	8	R/W	0x00	16.4.3/16-5
Supervisor/User Access Registers					
0xFC09_0004 0xFC09_4004	EPORT Data Register (EPDR*n*)	8	R/W	0xFF	16.4.4/16-5
0xFC09_0005 0xFC09_4005	EPORT Pin Data Register (EPPDR*n*)	8	R	See Section	16.4.5/16-5
0xFC09_0006 0xFC09_4006	EPORT Flag Register (EPFR*n*)	8	R/W	0x00	16.4.6/16-6



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

16.4.1 EPORT Pin Assignment Register (EPPAR)

The EPORT pin assignment register (EPPAR) controls the function of each pin individually.

Address: 0xFC09_0000 (EPPAR0)
0xFC09_4000 (EPPAR1)

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 16-2. EPORT Pin Assignment Register (EPPAR)

Table 16-3. 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

16.4.2 EPORT Data Direction Register (EPDDR)

The EPORT data direction register (EPDDR) controls the direction of each one of the pins individually.

Address: 0xFC09_0002 (EPDDR0)
0xFC09_4002 (EPDDR1)

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 16-3. EPORT Data Direction Register (EPDDR)

Table 16-4. 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

16.4.3 Edge Port Interrupt Enable Register (EPIER)

The EPORT interrupt enable register (EPIER) enables interrupt requests for each pin individually.

Address: 0xFC09_0003 (EPIER0)
0xFC09_4003 (EPIER1)

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 16-4. EPORT Port Interrupt Enable Register (EPIER)

Table 16-5. 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

16.4.4 Edge Port Data Register (EPDR)

The EPORT data register (EPDR) holds the data to be driven to the pins.

Address: 0xFC09_0004 (EPDR0)
0xFC09_4004 (EPDR1)

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 16-5. EPORT Port Data Register (EPDR)

Table 16-6. 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.

16.4.5 Edge Port Pin Data Register (EPPDR)

The EPORT pin data register (EPPDR) reflects the current state of the pins.

Address: 0xFC09_0005 (EPPDR0)
0xFC09_4005 (EPPDR1)

Access: User read-only

	7	6	5	4	3	2	1	0
R	EPPD7	EPPD6	EPPD5	EPPD4	EPPD3	EPPD2	EPPD1	EPPD0
W								
Reset:	[$\overline{\text{IRQ7}}$]	[$\overline{\text{IRQ6}}$]	[$\overline{\text{IRQ5}}$]	[$\overline{\text{IRQ4}}$]	[$\overline{\text{IRQ3}}$]	[$\overline{\text{IRQ2}}$]	[$\overline{\text{IRQ1}}$]	[$\overline{\text{IRQ0}}$]

Figure 16-6. EPORT Port Pin Data Register (EPPDR)

Table 16-7. EPPDR Field Descriptions

Field	Description
7–0 EPPD*n*	Edge port pin data bits. The read-only EPPDR reflects the current state of the EPORT pins $\overline{\text{IRQ7}}$ – $\overline{\text{IRQ0}}$. Writing to EPPDR has no effect, and the write cycle terminates normally. Reset does not affect EPPDR.

16.4.6 Edge Port Flag Register (EPFR)

The EPORT flag register (EPFR) individually latches EPORT edge events.

Address: 0xFC09_0006 (EPFR0)
0xFC09_4006 (EPFR1)

Access: User read/write

	7	6	5	4	3	2	1	0
R	EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	EPF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 16-7. EPORT Port Flag Register (EPFR)

Table 16-8. EPFR Field Descriptions

Field	Description
7–0 EPF*n*	Edge port flag bits. When an EPORT pin is configured for edge triggering, its corresponding read/write bit in EPFR indicates that the selected edge has been detected. Reset clears EPF7 – EPF0. Bits in this register are set when the selected edge is detected on the corresponding pin. A bit remains set until cleared by writing a 1 to it. Writing 0 has no effect. If a pin is configured as level-sensitive (EPPAR*n* = 00), pin transitions do not affect this register. 0 Selected edge for $\overline{\text{IRQ}n}$ pin not detected 1 Selected edge for $\overline{\text{IRQ}n}$ pin detected

Chapter 17
Enhanced Direct Memory Access (eDMA)

17.1 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.

17.1.1 Block Diagram

Figure 17-1 is a block diagram of the eDMA module.

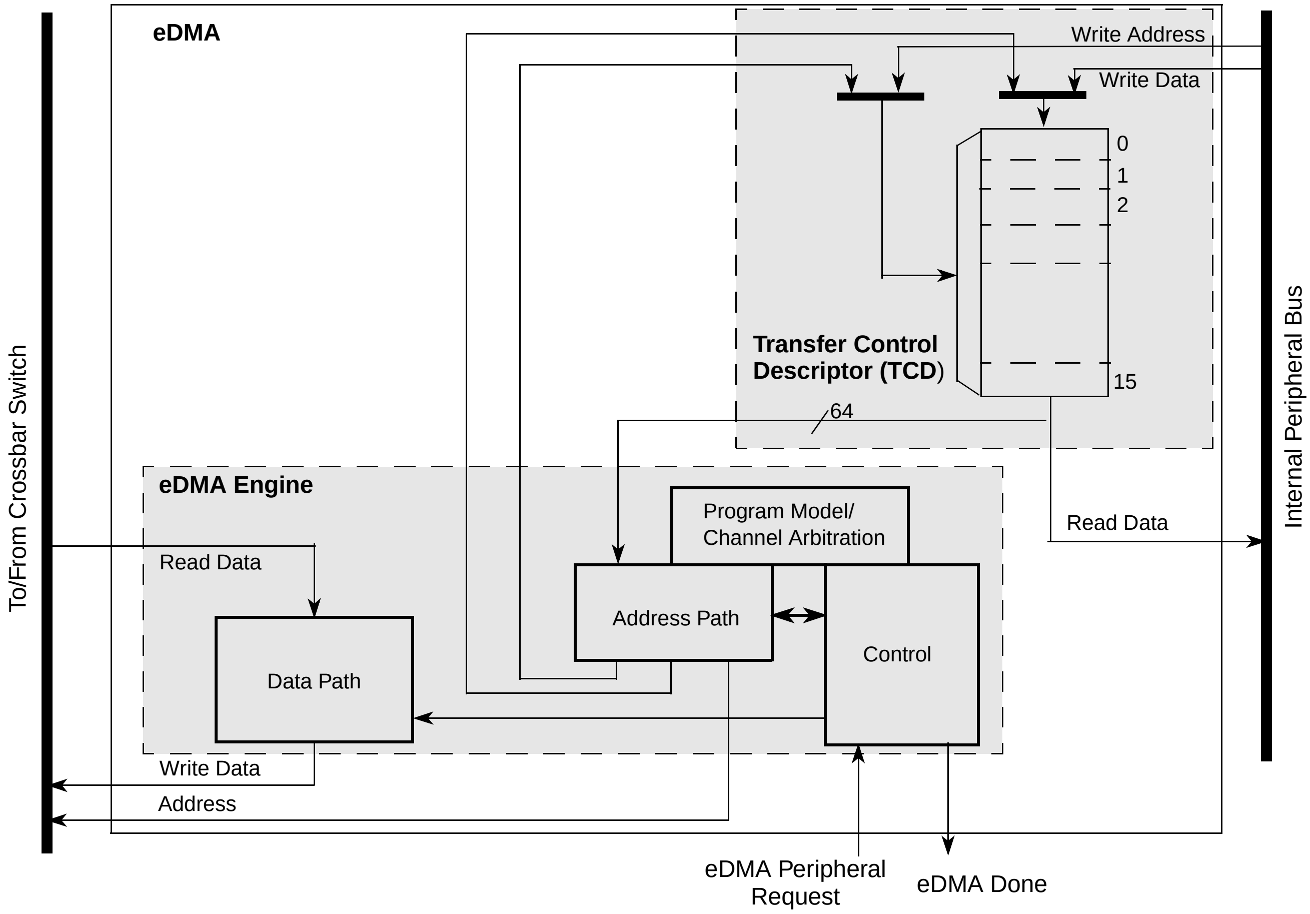


Figure 17-1. eDMA Block Diagram

17.1.2 Features

The eDMA is a highly-programmable data-transfer engine optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known and not defined within the data packet itself. The eDMA module features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source and destination addresses and transfer size, plus support for enhanced addressing modes
- 16-channel implementation that performs complex data transfers with minimal intervention from a host processor
  - Internal data buffer, used as temporary storage to support 16-byte burst transfers
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD stored in local memory for each channel
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continual transfers
  - Peripheral-paced hardware requests (one per channel)
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel and logically summed together to form one error interrupt to the interrupt controller
- Optional support for scatter/gather DMA processing

Throughout this chapter, *n* is used to reference the channel number.

17.2 Modes of Operation

17.2.1 Normal Mode

In normal mode, the eDMA transfers data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

A service request initiates a transfer of a specific number of bytes (NBYTES) as specified in the transfer control descriptor (TCD). The minor loop is the sequence of read-write operations that transfers these NBYTES per service request. A major loop is the number of minor loop iterations defining a task.

17.2.2 Debug Mode

In debug mode, the eDMA stops transferring data. If debug mode is entered during the transfer of a data block described by a minor loop in the current active channel's TCD, the eDMA continues operation until completion of the minor loop.

17.3 External Signal Description

This section describes the external signals of the eDMA controller.

Table 17-1. 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.

17.3.1 External Signal Timing

Asserting the external DMA request signal, $\overline{\text{DREQ}n}$, initiates a service request for that channel. It must remain asserted until the corresponding $\overline{\text{DACK}n}$ signal indicates the channel's data transfer has started. The $\overline{\text{DACK}n}$ output is asserted for one cycle during the address phase of the channel's first internal read access.

- When no further requests are needed, the $\overline{\text{DREQ}n}$ signal must negate after the $\overline{\text{DACK}n}$ assertion and on or before the second cycle following the data phase of the last internal bus write (see Figure 17-2).
- 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.

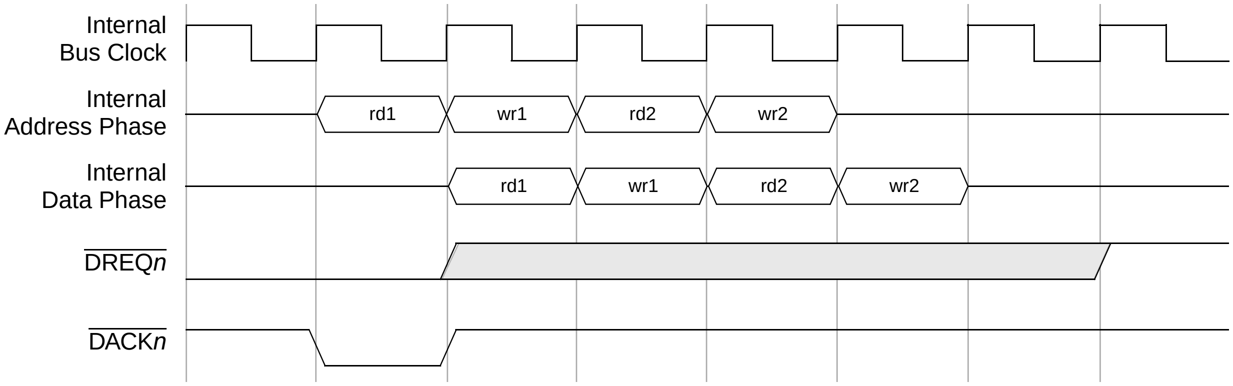


Figure 17-2. $\overline{\text{DREQ}n}$ and $\overline{\text{DACK}n}$ Timing

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.

17.4 Memory Map/Register Definition

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions, while the second region corresponds to the local transfer control descriptor memory.

Reading reserved bits in a register return the value of zero and writes to reserved bits in a register are ignored. Reading or writing to a reserved memory location generates a bus error.

Table 17-2. 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	17.4.1/17-4
0xFC04_4004	eDMA Error Status Register (EDMA_ES)	32	R	0x0000_0000	17.4.2/17-5
0xFC04_400E	eDMA Enable Request Register (EDMA_ERQ)	16	R/W	0x0000	17.4.3/17-8
0xFC04_4016	eDMA Enable Error Interrupt Register (EDMA_EEI)	16	R/W	0x0000	17.4.4/17-9
0xFC04_4018	eDMA Set Enable Request (EDMA_SERQ)	8	W	0x00	17.4.5/17-10
0xFC04_4019	eDMA Clear Enable Request (EDMA_CERQ)	8	W	0x00	17.4.6/17-10
0xFC04_401A	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	8	W	0x00	17.4.7/17-11
0xFC04_401B	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	8	W	0x00	17.4.8/17-11
0xFC04_401C	eDMA Clear Interrupt Request Register (EDMA_CINT)	8	W	0x00	17.4.9/17-12
0xFC04_401D	eDMA Clear Error Register (EDMA_CERR)	8	W	0x00	17.4.10/17-13
0xFC04_401E	eDMA Set START Bit Register (EDMA_SSRT)	8	W	0x00	17.4.11/17-13
0xFC04_401F	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	8	W	0x00	17.4.12/17-14
0xFC04_4026	eDMA Interrupt Request Register (EDMA_INT)	32	R/W	0x0000	17.4.13/17-15
0xFC04_402E	eDMA Error Register (EDMA_ERR)	32	R/W	0x0000	17.4.14/17-15
0xFC04_4100 + hex(*n*)	eDMA Channel *n* Priority Register (DCHPRI*n*) for *n* = 0 – 15	8	R/W	See Section	17.4.15/17-16
0xFC04_5000 + hex(32×*n*)	Transfer Control Descriptor (TCD*n*) for *n* = 0 – 15	256	R/W	See Section	17.4.16/17-17

17.4.1 eDMA Control Register (EDMA_CR)

The EDMA_CR defines the basic operating configuration of the eDMA. Arbitration can be configured to use a fixed-priority or a round-robin scheme. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The channel priority registers assign the priorities (see Section 17.4.15, "eDMA Channel n Priority Registers (DCHPRIn)"). In round-robin arbitration mode, the channel priorities are ignored, and channels are cycled through without regard to priority.

NOTE

For proper operation, writes to the EDMA_CR register must only be performed when the DMA channels are inactive (TCR*n*_CSR[ACTIVE] bits are cleared).

Address: 0xFC04_4000 (EDMA_CR) Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-3. eDMA Control Register (EDMA_CR)

Table 17-3. EDMA_CR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–3	Reserved, should be cleared.
2 ERCA	Enable round robin channel arbitration. 0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
1 EDBG	Enable debug. 0 When in debug mode the DMA continues to operate. 1 When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when the system exits debug mode or the EDBG bit is cleared.
0	Reserved, must be cleared.

17.4.2 eDMA Error Status Register (EDMA_ES)

The EDMA_ES provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer-control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is reported when the starting source or destination address, source or destination offsets, minor loop byte count, or the transfer size represent an inconsistent state. Each of these possible causes are detailed in the below list:

- The addresses and offsets must be aligned on 0-modulo-transfer-size boundaries
- The minor loop byte count must be a multiple of the source and destination transfer sizes.
- All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

- In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal. All channel priority levels must be unique when fixed arbitration mode is enabled.
- If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (DLAST_SGA) is not aligned on a 32-byte boundary.
- If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD*n*_CITER[E_LINK] bit does not equal the TCD*n*_BITER[E_LINK] bit.

If enabled, all configuration error conditions, except the scatter/gather and minor-loop link errors, report as the channel activates and asserts an error interrupt request. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system-bus error occurs, the channel terminates after the read or write transaction (which is already pipelined after errant access) has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel terminates due to the destination bus error.

The occurrence of any error causes the eDMA engine to stop the active channel immediately, and the appropriate channel bit in the eDMA error register is asserted. At the same time, the details of the error condition are loaded into the EDMA_ES. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the eDMA engine continues operating by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel executes and terminate with the same error condition.

Address: 0xFC04_4004 (EDMA_ES)   Access: User read-only

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	CPE	0	0	ERRCHN				SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-4. eDMA Error Status Register (EDMA_ES)

Table 17-4. 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 TCD*n*_SADDR field. TCD*n*_SADDR is inconsistent with TCD*n*_ATTR[SSIZE]
6 SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_SOFF field. TCD*n*_SOFF is inconsistent with TCD*n*_ATTR[SSIZE].
5 DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_DADDR field. TCD*n*_DADDR is inconsistent with TCD*n*_ATTR[DSIZE].
4 DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_DOFF field. TCD*n*_DOFF is inconsistent with TCD*n*_ATTR[DSIZE].
3 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_NBYTES or TCD*n*_CITER fields. • TCD*n*_NBYTES is not a multiple of TCD*n*_ATTR[SSIZE] and TCD*n*_ATTR[DSIZE], or • TCD*n*_CITER[CITER] is equal to zero, or • TCD*n*_CITER[E_LINK] is not equal to TCD*n*_BITER[E_LINK].
2 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD*n*_DLAST_SGA field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD*n*_CSR[E_SG] is enabled. TCD*n*_DLAST_SGA is not on a 32 byte boundary.
1 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
0 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

17.4.3 eDMA Enable Request Register (EDMA_ERQ)

The EDMA_ERQ register provides a bit map for the 16 implemented channels to enable the request signal for each channel. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the EDMA_SERQ and EDMA_CERQ. The EDMA_{S,C}ERQR are provided so the request enable for a single channel can easily be modified without needing to perform a read-modify-write sequence to the EDMA_ERQ.

DMA request input signals and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made explicitly through software or a linked channel request.

Address: 0xFC04_400E (EDMA_ERQ) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 9	ERQ 8	ERQ 7	ERQ 6	ERQ 5	ERQ 4	ERQ 3	ERQ 2	ERQ 1	ERQ 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-5. eDMA Enable Request Register (EDMA_ERQ)

Table 17-5. 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 17-6.

Table 17-6. DMA Request Summary for eDMA

Channel	Source	Description
0	$\overline{\text{DREQ0}}$	External DMA request 0
1	$\overline{\text{DREQ1}}$	External DMA request 1
2	UISR0[FFULL/RXRDY]	UART0 Receive
3	UISR0[TXRDY]	UART0 Transmit
4	UISR1[FFULL/RXRDY]	UART1 Receive
5	UISR1[TXRDY]	UART1 Transmit
6	UISR2[FFULL/RXRDY]	UART2 Receive
7	UISR2[TXRDY]	UART2 Transmit
8	DTER0[CAP] or DTER0[REF] / SSISR[RFF0]	Timer 0 / SSI0 Receive[1]
9	DTER1[CAP] or DTER1[REF] / SSISR[RFF1]	Timer 1 / SSI1 Receive[1]

Table 17-6. DMA Request Summary for eDMA (continued)

Channel	Source	Description
10	DTER2[CAP] or DTER2[REF] / SSISR[TFE0]	Timer 2 / SSI0 Transmit[1]
11	DTER3[CAP] or DTER3[REF] / SSISR[TFE1]	Timer 3 / SSI1 Transmit[1]
12	DSPI_SR[RFDF]	DSPI Receive
13	DSPI_SR[TFFF]	DSPI Transmit
14	—	Reserved/Not used
15	SDHC_SR[BRRDY, BWRDY]	SDHC

[1] For information on how to select between SSI and Timer sources, refer to Chapter 9, “Chip Configuration Module (CCM).”

As a given channel completes the processing of its major iteration count, a flag in the transfer control descriptor that affect the ending state of the EDMA_ERQ bit for that channel. If the TCD*n*_CSR[D_REQ] bit is set, the corresponding EDMA_ERQ bit is cleared, disabling the DMA request. If the D_REQ bit clears, the state of the EDMA_ERQ bit is unaffected.

17.4.4 eDMA Enable Error Interrupt Registers (EDMA_EEI)

The EDMA_EEI register provides a bit map for the 16 channels to enable the error interrupt signal for each channel. The state of any given channel’s error interrupt enable is directly affected by writes to this register; it is also affected by writes to the EDMA_SEEI and EDMA_CEEI. The EDMA_{S,C}EEIR are provided so the error interrupt enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_EEI register.

The DMA error indicator and the error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted to the interrupt controller.

Address: 0xFC04_4016 (EDNA_EEI) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI9	EEI8	EEI7	EEI6	EEI5	EEI4	EEI3	EEI2	EEI1	EEI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-6. eDMA Enable Error Interrupt Register (EDMA_EEI)

Table 17-7. 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.

17.4.5 eDMA Set Enable Request Register (EDMA_SERQ)

The EDMA_SERQ provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQ to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQ to be set. Setting the SAER bit provides a global set function, forcing the entire contents of EDMA_ERQ to be set. Reads of this register return all zeroes.

Address: 0xFC04_4018 (EDMA_SERQ) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAER			SERQ			
Reset	0	0	0	0	0	0	0	0

Figure 17-7. eDMA Set Enable Request Register (EDMA_SERQ)

Table 17-8. 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

17.4.6 eDMA Clear Enable Request Register (EDMA_CERQ)

The EDMA_CERQ provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQ to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQ to be cleared. Setting the CAER bit provides a global clear function, forcing the entire contents of the EDMA_ERQ to be cleared, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: 0xFC04_4019 (EDMA_CERQ) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAER			CERQ			
Reset	0	0	0	0	0	0	0	0

Figure 17-8. eDMA Clear Enable Request Register (EDMA_CERQ)

Table 17-9. EDMA_CERQ Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 CAER	Clear all enable requests. 0 Clear only those EDMA_ERQ bits specified in the CERQ field. 1 Clear all bits in EDMA_ERQ.
5–4	Reserved, must be cleared.
3–0 CERQ	Clear enable request. Clears the corresponding bit in EDMA_ERQ.

17.4.7 eDMA Set Enable Error Interrupt Register (EDMA_SEEI)

The EDMA_SEEI provides a simple memory-mapped mechanism to set a given bit in the EDMA_EEI to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEI to be set. Setting the SAEE bit provides a global set function, forcing the entire EDMA_EEI contents to be set. Reads of this register return all zeroes.

Address: 0xFC04_401A (EDMA_SEEI) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAEE			SEEI			
Reset	0	0	0	0	0	0	0	0

Figure 17-9. eDMA Set Enable Error Interrupt Register (EDMA_SEEI)

Table 17-10. 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.

17.4.8 eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)

The EDMA_CEEI provides a simple memory-mapped mechanism to clear a given bit in the EDMA_EEI to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEI to be cleared. Setting the CAEE bit provides a global clear function, forcing the EDMA_EEI contents to be cleared, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: 0xFC04_401B (EDNA_CEEI) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAEE			CEEI			
Reset	0	0	0	0	0	0	0	0

Figure 17-10. eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)

Table 17-11. 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.

17.4.9 eDMA Clear Interrupt Request Register (EDMA_CINT)

The EDMA_CINT provides a simple, memory-mapped mechanism to clear a given bit in the EDMA_INT to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_INT to be cleared. Setting the CAIR bit provides a global clear function, forcing the entire contents of the EDMA_INT to be cleared, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Address: 0xFC04_401C (EDMA_CINT) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAIR			CINT			
Reset	0	0	0	0	0	0	0	0

Figure 17-11. eDMA Clear Interrupt Request (EDMA_CINT)

Table 17-12. 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 17-12. 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.

17.4.10 eDMA Clear Error Register (EDMA_CERR)

The EDMA_CERR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERR to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERR to be cleared. Setting the CAEI bit provides a global clear function, forcing the EDMA_ERR contents to be cleared, clearing all channel error indicators. Reads of this register return all zeroes.

Address: 0xFC04_401D (EDMA_CERR) Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAEI			CERR			
Reset	0	0	0	0	0	0	0	0

Figure 17-12. eDMA Clear Error Register (EDMA_CERR)

Table 17-13. 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.

17.4.11 eDMA Set START Bit Register (EDMA_SSRT)

The EDMA_SSRT provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting the SAST bit provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Address: 0xFC04_401E (EDMA_SSRT)   Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAST			SSRT			
Reset	0	0	0	0	0	0	0	0

Figure 17-13. eDMA Set START Bit Register (EDMA_SSRT)

Table 17-14. 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].

17.4.12 eDMA Clear DONE Status Bit Register (EDMA_CDNE)

The EDMA_CDNE provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting the CADN bit provides a global clear function, forcing all DONE bits to be cleared. Reads of this register return all zeroes.

Address: 0xFC04_401F (EDMA_CDNE)   Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W		CADN			CDNE			
Reset	0	0	0	0	0	0	0	0

Figure 17-14. eDMA Clear DONE Status Bit Register (EDMA_CDNE)

Table 17-15. 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].

17.4.13 eDMA Interrupt Request Register (EDMA_INT)

The EDMA_INT provide a bit map for the 16 channels signaling the presence of an interrupt request for each channel. Depending on the appropriate bit setting in the transfer-control descriptions, the eDMA engine generates an interrupt a data transfer completion. The outputs of this register are directly routed to the interrupt controller (INTC). During the interrupt-service routine associated with any given channel, it is the software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CINT in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CINT. On writes to the EDMA_INT, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CINT is provided so the interrupt request for a single channel can easily be cleared without the need to perform a read-modify-write sequence to the EDMA_INT.

Address: 0xFC04_4026 (EDMA_INT) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	INT15	INT14	INT13	INT12	INT11	INT10	INT9	INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-15. eDMA Interrupt Request Register (EDMA_INT)

Table 17-16. 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.

17.4.14 eDMA Error Register (EDMA_ERR)

The EDMA_ERR provide a bit map for the 16 channels, signaling the presence of an error for each channel. The eDMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEI,and then routed to the interrupt controller. During the execution of the interrupt-service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error-interrupt request. Typically, a write to the EDMA_CERR in the interrupt-service routine is used for this purpose. The normal DMA channel completion indicators (setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request) are not affected when an error is detected.

The contents of this register can also be polled because a non-zero value indicates the presence of a channel error regardless of the state of the EDMA_EEI. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_CERR. On writes to the EDMA_ERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The EDMA_CERR is provided so the error indicator for a single channel can easily be cleared.

Address: 0xFC04_402E (EDMA_ERR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 9	ERR 8	ERR 7	ERR 6	ERR 5	ERR 4	ERR 3	ERR 2	ERR 1	ERR 0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-16. eDMA Error Register (EDMA_ERR)

Table 17-17. 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.

17.4.15 eDMA Channel *n* Priority Registers (DCHPRI*n*)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; for example, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. Software must program the channel priorities with unique values. Otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per-channel basis by setting the DCHPRI*n*[ECP] bit. Channel preemption allows the executing channel’s data transfers to temporarily suspend in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected.

Address: 0xFC04_4100 + *n*, where *n* = 0 – 15 (DCHPRI*n*) Access: User read/write

	7	6	5	4	3	2	1	0
R	ECP	0	0	0	CHPRI			
W								
Reset	0	0	0	0	–[1]	–[1]	–[1]	–[1]

[1] Reset value for the channel priority fields, CHPRI, is equal to the corresponding channel number for each priority register, i.e., DCHPRI15[CHPRI] equals 0b1111.

Figure 17-17. eDMA Channel *n* Priority Register (DCHPRI*n*)

Table 17-18. 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.

17.4.16 Transfer Control Descriptors (TCD*n*)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. Each TCD*n* definition is presented as 11 registers of 16 or 32 bits. Table 17-19 is a register list of the basic TCD structure.

Table 17-19. 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 17-18. TCD*n* Source Address (TCD*n*_SADDR)

Table 17-20. 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 17-19. TCD*n* Transfer Attributes (TCD*n*_ATTR)

Table 17-21. 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 17-20. TCDn Signed Source Address Offset (TCD*n*_SOFF)

Table 17-22. 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 W	NBYTES																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 17-21. TCD*n* Minor Byte Count (TCD*n*_NBYTES)

Table 17-23. 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 W	SLAST																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 17-22. TCD*n* Source Last Address Adjustment (TCD*n*_SLAST)

Table 17-24. 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 W	DADDR																															
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 17-23. TCD*n* Destination Address (TCD*n*_DADDR)

Table 17-25. 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 17-24. TCD*n* Current Major Iteration Count (TCD*n*_CITER)

Table 17-26. 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.

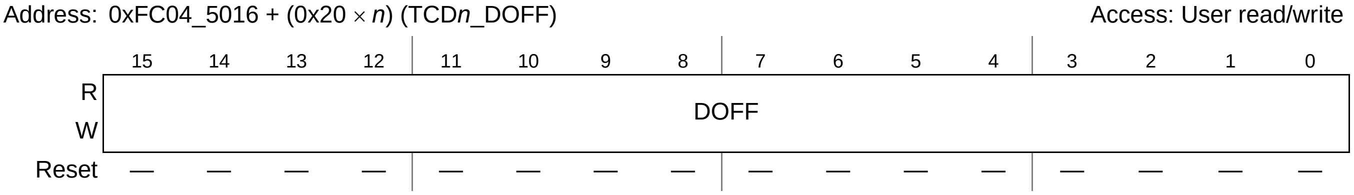

Address: 0xFC04_5016 + (0x20 × n) (TCDn_DOFF)
Access: User read/write
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
R
W
DOFF
Reset — — — — — — — — — — — — — — — —

Figure 17-25. TCD*n* Destination Address Signed Offset (TCD*n*_DOFF)

Table 17-27. TCD*n*_DOFF Field Descriptions

Field	Description
15–0 DOFF	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.

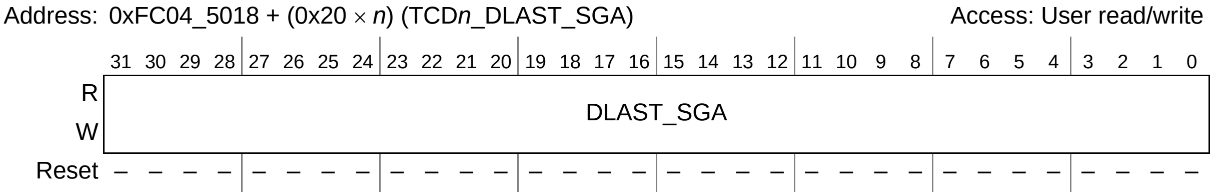

Address: 0xFC04_5018 + (0x20 × n) (TCDn_DLAST_SGA)
Access: User read/write
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
R
W
DLAST_SGA
Reset –

Figure 17-26. TCD*n* Destination Last Address Adjustment (TCD*n*_DLAST_SGA)

Table 17-28. TCD*n*_DLAST_SGA Field Descriptions

Field	Description
31–0 DLAST_SGA	Destination last address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If (TCD*n*_CSR[E_SG] = 0) then • Adjustment value added to the destination address at the completion of the major iteration count. This value can apply to restore the destination address to the initial value or adjust the address to reference the next data structure. else • This address points to the beginning of a 0-modulo-32-byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32-byte, else a configuration error is reported.

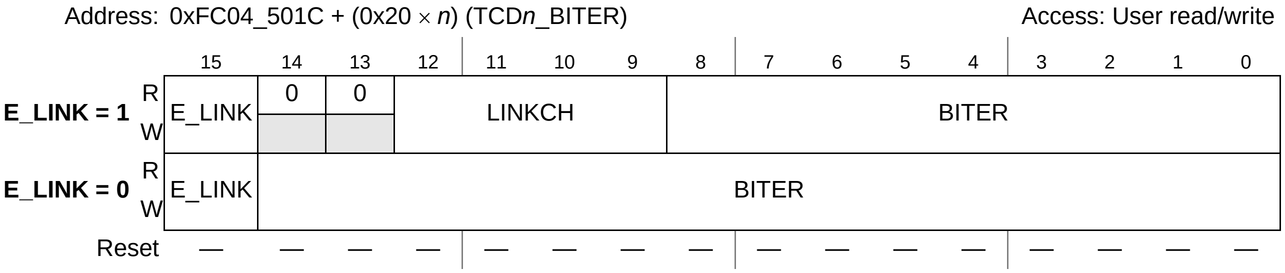


Figure 17-27. TCD*n* Beginning Major Iteration Count (TCD*n*_BITER)

Table 17-29. 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 17-28. TCD*n* Control and Status (TCD*n*_CSR)

Table 17-30. TCD*n*_CSR Field Descriptions

Field	Description
15–14 BWC	Bandwidth control. Throttles the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the crossbar switch (XBS). 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for 4 cycles after each r/w 11 eDMA engine stalls for 8 cycles after each r/w **Note:** If the source and destination sizes are equal, this field is ignored between the first and second transfers and after the last write of each minor loop. This behavior is a side effect of reducing start-up latency.
13–12	Reserved, must be cleared.
11–8 MAJOR_LINKCH	Link channel number. If (MAJOR_E_LINK = 0) then • No channel-to-channel linking (or chaining) is performed after the major loop counter is exhausted. else • After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these four bits by setting that channel's TCD*n*_CSR[START] bit. 0–15 Link to DMA channel 0–15
7 DONE	Channel done. This flag indicates the eDMA has completed the major loop. The eDMA engine sets it as the CITER count reaches zero; The software clears it, or the hardware when the channel is activated. **Note:** This bit must be cleared to write the MAJOR_E_LINK or E_SG bits.
6 ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and the eDMA clears it as the minor loop completes or if any error condition is detected.
5 MAJOR_E_LINK	Enable channel-to-channel linking on major loop complete. As the channel completes the major loop, this flag enables the linking to another channel, defined by MAJOR_LINKCH. The link target channel initiates a channel service request via an internal mechanism that sets the TCD*n*_CSR[START] bit of the specified channel. **Note:** To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD*n*_CSR[DONE] bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
4 E_SG	Enable scatter/gather processing. As the channel completes the major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure loaded as the transfer control descriptor into the local memory. **Note:** To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD*n*_CSR[DONE] bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the major loop completes its execution.
3 D_REQ	Disable request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQ bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQ bit is not affected. 1 The channel's EDMA_ERQ bit is cleared when the major loop is complete.

Table 17-30. 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.

17.5 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

17.5.1 eDMA Microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer-control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules:

- eDMA Engine
  - — Address Path:

    This block implements registered versions of two channel transfer control descriptors, channel x and channel y, and manages all master bus-address calculations. All the channels provide the same functionality. This structure allows data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel activation is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed, unless preempted by a higher priority channel. This provides a mechanism (enabled by DCHPRI*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

    When any channel is selected to execute, the contents of its TCD are read from local memory and loaded into the address path channel x registers for a normal start and into channel y registers for a preemption start. After the minor loop completes execution, the address path hardware writes the new values for the TCD*n*_{SADDR, DADDR, CITER} back to local memory. If the major iteration count is exhausted, additional processing are performed, including the final address pointer updates, reloading the TCD*n*_CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter/gather operation.

- Data Path:

  This block implements the bus master read/write datapath. It includes 16 bytes of register storage and the necessary multiplex logic to support any required data alignment. The internal read data bus is the primary input, and the internal write data bus is the primary output.

  The address and data path modules directly support the 2-stage pipelined internal bus. The address path module represents the 1st stage of the bus pipeline (address phase), while the data path module implements the 2nd stage of the pipeline (data phase).

- Program Model/Channel Arbitration:

  This block implements the first section of the eDMA programming model as well as the channel arbitration logic. The programming model registers are connected to the internal peripheral bus (not shown). The eDMA peripheral request inputs and interrupt request outputs are also connected to this block (via control logic).

- Control:

  This block provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read/destination write operations until the number of bytes specified in the minor loop byte count has moved. For descriptors where the sizes are not equal, multiple accesses of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- Transfer Control Descriptor Memory
  - Memory Controller:

    This logic implements the required dual-ported controller, managing accesses from the eDMA engine as well as references from the internal peripheral bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the peripheral transaction is stalled.

  - Memory Array: TCD storage is implemented using a single-port, synchronous RAM array.

17.5.2 eDMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 17-29, 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.

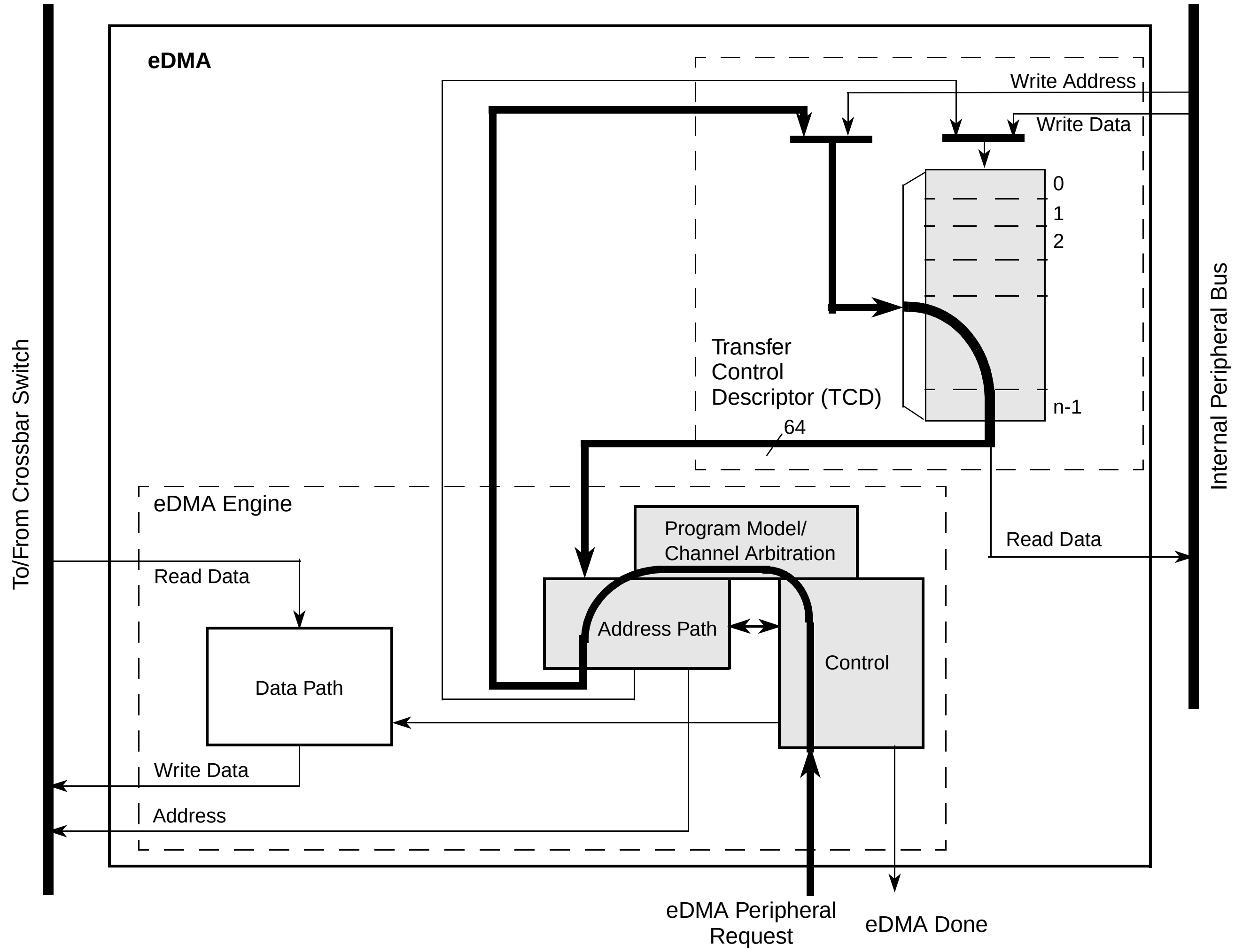


Figure 17-29. eDMA Operation, Part 1

In the second part of the basic data flow (Figure 17-30), 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.

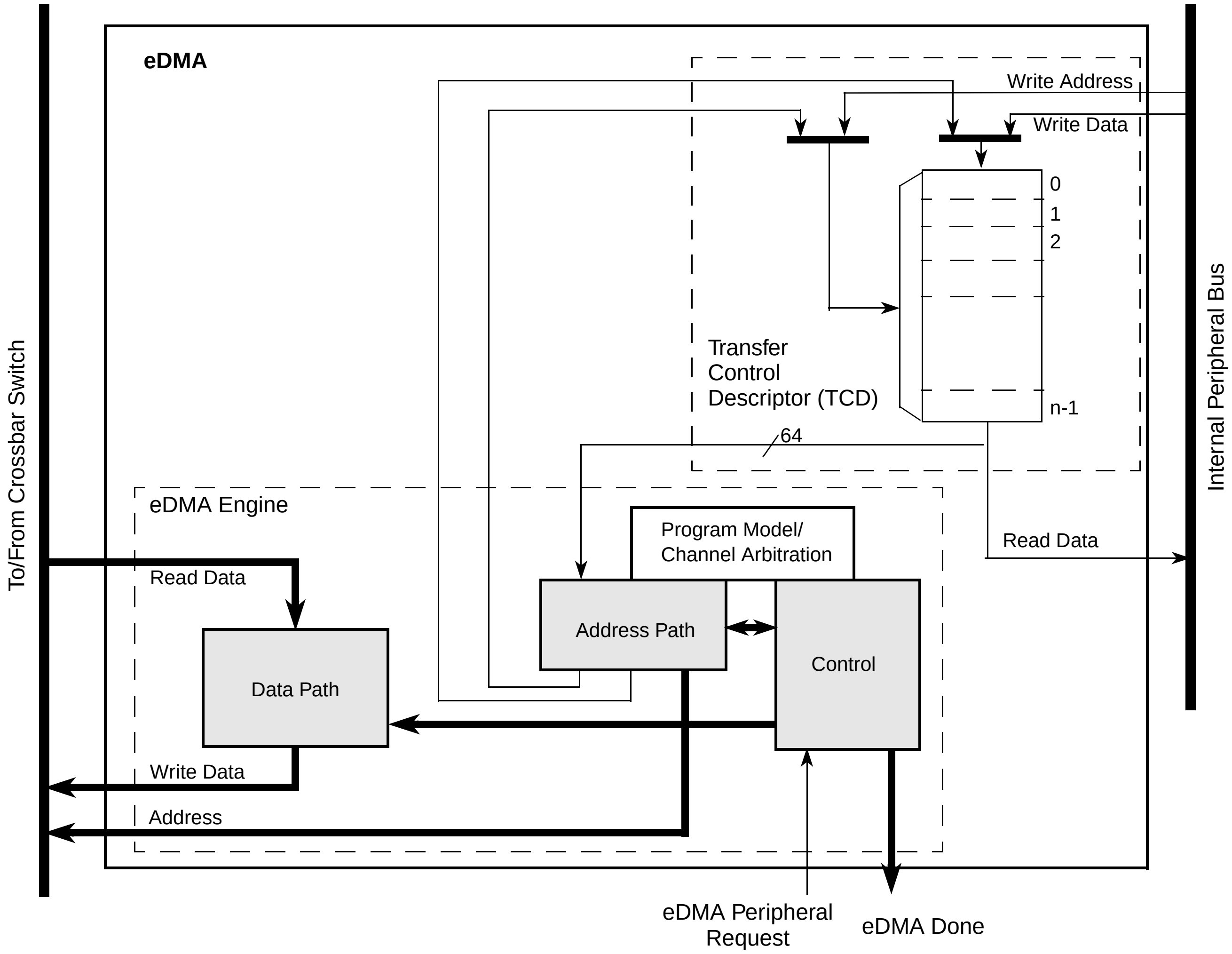


Figure 17-30. eDMA Operation, Part 2

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 17-31.

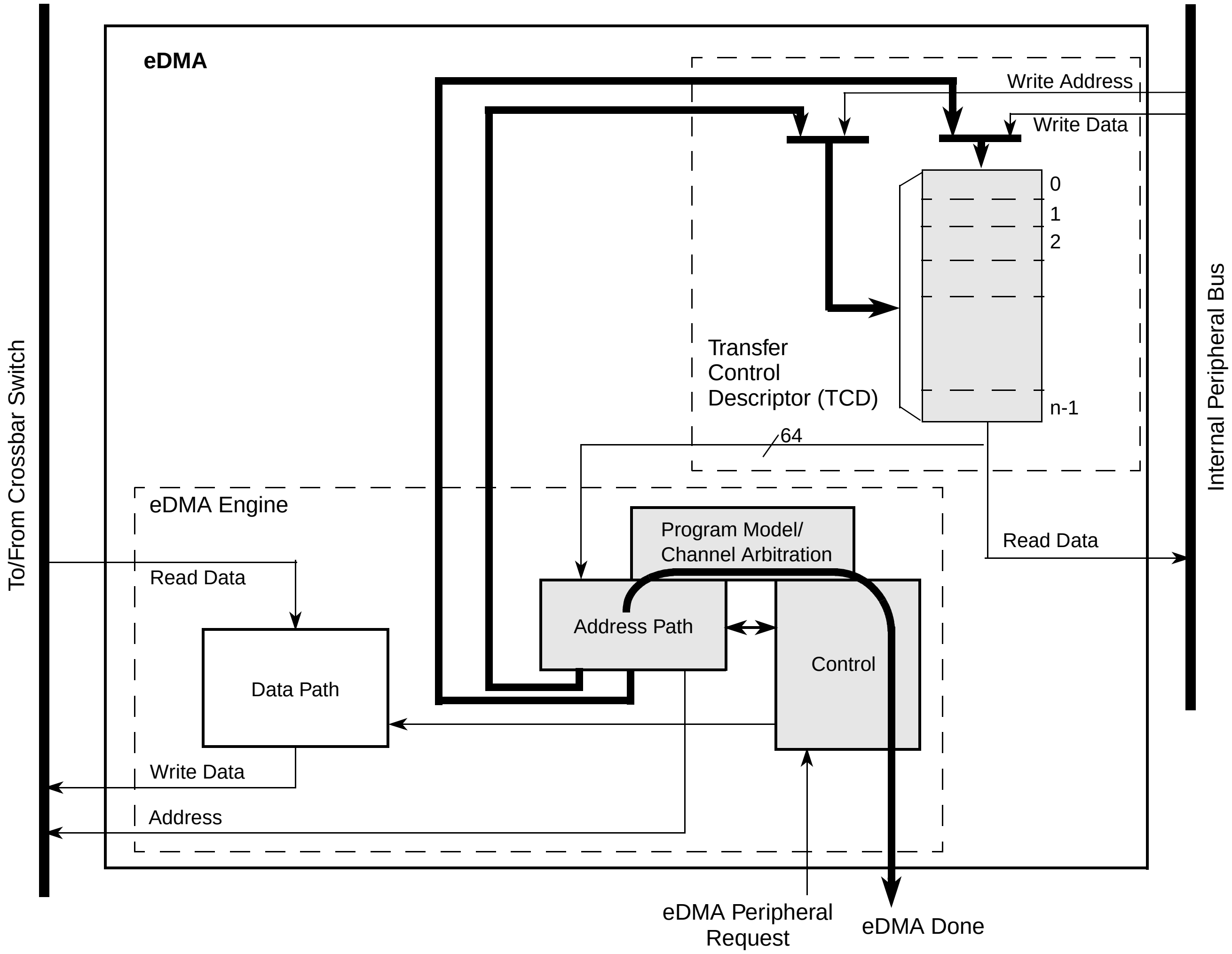


Figure 17-31. eDMA Operation, Part 3

17.6 Initialization/Application Information

17.6.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRI*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEI if so desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_ERQ.
6. Request channel service by software (setting the TCD*n*_CSR[START] bit) or hardware (slave device asserting its eDMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine read the entire TCD, including the TCD control and status fields (Table 17-31) 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 17-31. 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 17-32 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 17-32. Example of Multiple Loop Iterations

				Current Major Loop Iteration Count (CITER)
DMA Request	· · ·	Minor Loop	Major Loop	3
DMA Request	· · ·	Minor Loop		2
DMA Request	· · ·	Minor Loop		1

Table 17-33 lists the memory array terms and how the TCD settings interrelate.

Table 17-33. 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	

17.6.2 DMA Programming Errors

The eDMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of channel priority error (EDMA_ES[CPE]).

For all error types other than channel priority error, the channel number causing the error is recorded in the EDMA_ES. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

If priority levels are not unique, when any channel requests service, a channel priority error is reported. The highest channel priority with an active request is selected, but the lowest numbered channel with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts, and error reporting is associated with the selected channel.

17.6.3 DMA Arbitration Mode Considerations

17.6.3.1 Fixed Channel Arbitration

In this mode, the channel service request from the highest priority channel is selected to execute.

17.6.3.2 Round Robin Channel Arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels

17.6.4 DMA Transfer

17.6.4.1 Single Request

To perform a simple transfer of n bytes of data with one activation, set the major loop to one (TCD*n*_CITER = TCD*n*_BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the TCD*n*_CSR[DONE] bit is set and an interrupt generates if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a longword-wide port located at 0x2000. The address offsets are programmed in increments to match the transfer size: one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

Example 17-1. 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.

17.6.4.2 Multiple Requests

Besides transferring 32 bytes via two hardware requests, the next example is the same as previous. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in EDMA_ERQ, the slave device initiates channel service requests.

```
TCDn_CITER = TCDn_BITER = 2
TCDn_SLAST = –32
TCDn_DLAST_SGA = –32
```

This would generate the following sequence of events:

1. First hardware (eDMA peripheral) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD*n*_CSR[DONE] = 0, TCD*n*_CSR[START] = 0, TCD*n*_CSR[ACTIVE] = 1.
4. eDMA engine reads: channel TCD*n* data from local memory to internal register file.
5. The source to destination transfers are executed as follows:

a) Read byte from location 0x1000, read byte from location 0x1001, read byte from 0x1002, read byte from 0x1003.

b) Write longword to location 0x2000 → first iteration of the minor loop.

c) Read byte from location 0x1004, read byte from location 0x1005, read byte from 0x1006, read byte from 0x1007.

d) Write longword to location 0x2004 → second iteration of the minor loop.

e) Read byte from location 0x1008, read byte from location 0x1009, read byte from 0x100A, read byte from 0x100B.

f) Write longword to location 0x2008 → third iteration of the minor loop.
g) Read byte from location 0x100C, read byte from location 0x100D, read byte from 0x100E, read byte from 0x100F.
h) Write longword to location 0x200C → last iteration of the minor loop.

6. eDMA engine writes: TCD*n*_SADDR = 0x1010, TCD*n*_DADDR = 0x2010, TCD*n*_CITER = 1.
7. eDMA engine writes: TCD*n*_CSR[ACTIVE] = 0.
8. The channel retires → one iteration of the major loop. The eDMA goes idle or services the next channel.
9. Second hardware (eDMA peripheral) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD*n*_CSR[DONE] = 0, TCD*n*_CSR[START] = 0, TCD*n*_CSR[ACTIVE] = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
    a) Read byte from location 0x1010, read byte from location 0x1011, read byte from 0x1012, read byte from 0x1013.
    b) Write longword to location 0x2010 → first iteration of the minor loop.
    c) Read byte from location 0x1014, read byte from location 0x1015, read byte from 0x1016, read byte from 0x1017.
    d) Write longword to location 0x2014 → second iteration of the minor loop.
    e) Read byte from location 0x1018, read byte from location 0x1019, read byte from 0x101A, read byte from 0x101B.
    f) Write longword to location 0x2018 → third iteration of the minor loop.
    g) Read byte from location 0x101C, read byte from location 0x101D, read byte from 0x101E, read byte from 0x101F.
    h) Write longword to location 0x201C → last iteration of the minor loop → major loop complete.
14. eDMA engine writes: TCD*n*_SADDR = 0x1000, TCD*n*_DADDR = 0x2000, TCD*n*_CITER = 2 (TCD*n*_BITER).
15. eDMA engine writes: TCD*n*_CSR[ACTIVE] = 0, TCD*n*_CSR[DONE] = 1, EDMA_INT[n] = 1.
16. The channel retires → major loop complete. The eDMA goes idle or services the next channel.

17.6.4.3 Modulo Feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of 2. MOD is a 5-bit field for the source and destination in the TCD, and it specifies which lower address bits increment from their original value after the address+offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 17-34 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 17-34. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

17.6.5 eDMA TCD*n* Status Monitoring

17.6.5.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first is to read the TCD*n*_CITER field and test for a change. (Another method may be extracted from the sequence shown below). The second method is to test the TCD*n*_CSR[START] bit and the TCD*n*_CSR[ACTIVE] bit. The minor-loop-complete condition is indicated by both bits reading zero after the TCD*n*_CSR[START] was set. Polling the TCD*n*_CSR[ACTIVE] bit may be inconclusive, because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

	TCD*n*_CSR bits			State
	START	ACTIVE	DONE	
1	1	0	0	Channel service request via software
2	0	1	0	Channel is executing
3a	0	0	0	Channel has completed the minor loop and is idle
3b	0	0	1	Channel has completed the major loop and is idle

The best method to test for minor-loop completion when using hardware (peripheral) initiated service requests is to read the TCD*n*_CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer’s model.

The TCD status bits execute the following sequence for a hardware-activated channel:

	TCD*n*_CSR bits			State
	START	ACTIVE	DONE	
1	0	0	0	Channel service request via hardware (peripheral request asserted)
2	0	1	0	Channel is executing
3a	0	0	0	Channel has completed the minor loop and is idle
3b	0	0	1	Channel has completed the major loop and is idle

For both activation types, the major-loop-complete status is explicitly indicated via the TCD*n*_CSR[DONE] bit.

The TCD*n*_CSR[START] bit is cleared automatically when the channel begins execution regardless of how the channel activates.

17.6.5.2 Active Channel TCD*n* Reads

The eDMA reads back the true TCD*n*_SADDR, TCD*n*_DADDR, and TCD*n*_NBYTES values if read while a channel executes. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine currently uses in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

17.6.5.3 Preemption Status

Preemption is available only when fixed arbitration is selected as the channel arbitration mode. A preemptive situation is one in which a preempt-enabled channel runs and a higher priority request becomes active. When the eDMA engine is not operating in fixed channel arbitration mode, the determination of the actively running relative priority outstanding requests become undefined. Channel priorities are treated as equal (constantly rotating) when round-robin arbitration mode is selected.

The TCD*n*_CSR[ACTIVE] bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one major loop iteration. If two TCD*n*_CSR[ACTIVE] bits are set simultaneously in the global TCD map, a higher priority channel is actively preempting a lower priority channel.

17.6.6 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD*n*_CSR[START] bit of another channel (or itself), therefore initiating a service request for that channel. When properly enabled, the EDMA engine automatically performs this operation at the major or minor loop completion.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD*n*_CITER[E_LINK] field determines whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, the initial fields of:

```
TCDn_CITER[E_LINK] = 1
TCDn_CITER[LINKCH] = 0xC
TCDn_CITER[CITER] value = 0x4
TCDn_CSR[MAJOR_E_LINK] = 1
TCDn_CSR[MAJOR_LINKCH] = 0x7
```

executes as:

1. Minor loop done → set TCD12_CSR[START] bit
2. Minor loop done → set TCD12_CSR[START] bit
3. Minor loop done → set TCD12_CSR[START] bit
4. Minor loop done, major loop done → set TCD7_CSR[START] bit

When minor loop linking is enabled (TCD*n*_CITER[E_LINK] = 1), the TCD*n*_CITER[CITER] field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled (TCD*n*_CITER[E_LINK] = 0), the TCD*n*_CITER[CITER] field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD*n*_CITER[LINKCH] field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

> The TCD*n*_CITER[E_LINK] bit and the TCD*n*_BITER[E_LINK] bit must equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, half-way done interrupt point.

Table 17-35 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 17-35. 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

17.6.7 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

17.6.7.1 Dynamic Channel Linking and Dynamic Scatter/Gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD*n*_CSR[MAJOR_E_LINK] or TCD*n*_CSR[E_SG] bits during channel execution. These bits are read from the TCD local memory at the end of channel execution, therefore allowing software to enable either feature during channel execution.

Because software can change the configuration during execution, a coherency sequence must be followed. Consider the scenario the user attempts to execute a dynamic channel link by enabling the TCD*n*_CSR[MAJOR_E_LINK] bit as the eDMA engine retires the channel. The TCD*n*_CSR[MAJOR_E_LINK] would be set in the programmer's model, but it would be indeterminate whether the actual link was made before the channel retired.

The following coherency sequence is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD*n*_CSR[MAJOR_E_LINK] bit.
2. Read back the TCD*n*_CSR[MAJOR_E_LINK] bit.
3. Test the TCD*n*_CSR[MAJOR_E_LINK] request status.
   a) If the bit is set, the dynamic link attempt was successful.
   b) If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD*n*_CSR[MAJOR_E_LINK] and TCD*n*_CSR[E_SG] bits to zero on any writes to a TCD*n* after the TCD*n*_CSR[DONE] bit for that channel is set, indicating the major loop is complete.

NOTE

> Software must clear the TCD*n*_CSR[DONE] bit before writing the TCD*n*_CSR[MAJOR_E_LINK] or TCD*n*_CSR[E_SG] bits. The TCD*n*_CSR[DONE] bit is cleared automatically by the eDMA engine after a channel begins execution.

Chapter 18
FlexBus

18.1 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, the data bus is switched dynamically between the SDRAM controller and the FlexBus module. However, when the SDRAM controller is in DDR mode, 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 setting. Take this into consideration throughout this chapter.

18.1.1 Overview

A multi-function external bus interface called the FlexBus interface controller is provided on the device with basic functionality of interfacing to slave-only devices. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry:

- External boot ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

For asynchronous devices, a simple chip-select based interface can be used.

The FlexBus interface has up to six general purpose chip-selects, $\overline{\text{FB\_CS}}$[5:0]. The actual number of chip selects available depends upon the device and its pin configuration. Chip-select $\overline{\text{FB\_CS0}}$ can be dedicated to boot memory access and programmed to be byte (8 bits), word (16 bits), or longword (32 bits) wide. Control signal timing is compatible with common ROM and flash memories.

18.1.2 Features

Key FlexBus features include:

- Six independent, user-programmable chip-select signals (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

18.2 External Signals

This section describes the external signals involved in data-transfer operations.

Table 18-1. 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
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 for more details.
FB_BE/BWE[3:0]	O	Byte enable/byte write enable
FB_OE	O	Output enable
FB_R/W	O	Read/write. 1 = Read, 0 = Write
FB_TS	O	Transfer start
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.

18.2.1 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.

18.2.2 Chip Selects (FB_CS[5:0])

The chip-select signal indicates which device is selected. A particular chip-select asserts when the transfer address is within the device’s address space, as defined in the base- and mask-address registers. The actual number of chip selects available depends upon the pin configuration.

18.2.3 Byte Enables/Byte Write Enables (FB_BE/BWE[3:0])

When driven low, the byte enable (FB_BE/BWE[3:0]) outputs indicate data is to be latched or driven onto a byte of the data bus. FB_BE/BWE*n* signals are asserted only to the memory bytes used during read or write accesses. A configuration option is provided to assert these signals on reads and writes (byte enable) or writes only (byte-write enable).

The FB_BE/BWE*n* signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM or cache. For external SRAM or flash devices, the FB_BE/BWE*n* outputs must be connected to individual byte strobe signals.

18.2.4 Output Enable (FB_OE)

The output enable signal (FB_OE) is sent to the interfacing memory and/or peripheral to enable a read transfer. FB_OE is only asserted during read accesses when a chip select matches the current address decode.

Because this device shares the FlexBus signals with the SDRAM controller, this signal tristates between bus cycles.

18.2.5 Read/Write (FB_R/W)

The processor drives the FB_R/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.

18.2.6 Transfer Start (FB_TS)

The assertion of FB_TS indicates that the device has begun a bus transaction and the address and attributes are valid. 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.

18.2.7 Transfer Acknowledge (FB_TA)

This signal indicates the external data transfer is complete. When the processor recognizes FB_TA during a read cycle, it latches the data and then terminates the bus cycle. When the processor recognizes FB_TA during a write cycle, the bus cycle is terminated.

If auto-acknowledge is disabled (CSCR*n*[AA] = 0), the external device drives $\overline{\text{FB\_TA}}$ to terminate the bus transfer; if auto-acknowledge is enabled (CSCR*n*[AA] = 1), $\overline{\text{FB\_TA}}$ is generated internally after a specified number of wait states, or the external device may assert external $\overline{\text{FB\_TA}}$ before the wait-state countdown, terminating the cycle early. The device negates $\overline{\text{FB\_CS}n}$ one cycle after the last $\overline{\text{FB\_TA}}$ asserts. During read cycles, the peripheral must continue to drive data until $\overline{\text{FB\_TA}}$ is recognized. For write cycles, the processor continues driving data one clock after $\overline{\text{FB\_CS}n}$ is negated.

The number of wait states is determined by CSCR*n* or the external $\overline{\text{FB\_TA}}$ input. If the external $\overline{\text{FB\_TA}}$ is used, the peripheral has total control on the number of wait states.

NOTE

External devices should only assert $\overline{\text{FB\_TA}}$ while the $\overline{\text{FB\_CS}n}$ signal to the external device is asserted.

Because this device shares the FlexBus signals with the SDRAM controller, this signal tristates between bus cycles.

18.3 Memory Map/Register Definition

The following tables describe the registers and bit meanings for configuring chip-select operation. Table 18-2 shows the chip-select register memory map.

The actual number of chip select registers available depends upon the device and its pin configuration. If the device does not support certain chip select signals or the pin is not configured for a chip-select function, then that corresponding set of chip-select registers has no effect on an external pin.

NOTE

You must set CSMR0[V] before the chip select registers take effect.

Table 18-2. FlexBus Chip Select Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC00_8000 + (*n* × 0xC)	Chip-Select Address Register (CSAR*n*) *n* = 0 – 5	32	R/W	0x0000_0000	18.3.1/18-5
0xFC00_8004 + (*n* × 0xC)	Chip-Select Mask Register (CSMR*n*) *n* = 0 – 5	32	R/W	0x0000_0000	18.3.2/18-5
0xFC00_8008 + (*n* × 0xC)	Chip-Select Control Register (CSCR*n*) *n* = 0 – 5	32	R/W	See Section	18.3.3/18-6

18.3.1 Chip-Select Address Registers (CSAR0 – CSAR5)

The CSAR*n* registers specify the chip-select base addresses.

NOTE

> Because the FlexBus module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x0000_0000 – 0x3FFF_FFFF and 0xC000_0000 – 0xDFFF_FFFF. Set the CSAR*n* registers appropriately.

Address: 0xFC00_8000 (CSAR0)
0xFC00_800C (CSAR1)
0xFC00_8018 (CSAR2)
0xFC00_8024 (CSAR3)
0xFC00_8030 (CSAR4)
0xFC00_803C (CSAR5)

Access: User read/write

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

Figure 18-1. Chip-Select Address Registers (CSAR*n*)

Table 18-3. CSAR*n* Field Descriptions

Field	Description
31–16 BA	Base address. Defines the base address for memory dedicated to chip-select $\overline{\text{FB\_CS}n}$. BA is compared to bits 31–16 on the internal address bus to determine if chip-select memory is being accessed.
15–0	Reserved, must be cleared.

18.3.2 Chip-Select Mask Registers (CSMR0 – CSMR5)

CSMR*n* registers specify the address mask and allowable access types for the respective chip-selects.

Address: 0xFC00_8004 (CSMR0)
0xFC00_8010 (CSMR1)
0xFC00_801C (CSMR2)
0xFC00_8028 (CSMR3)
0xFC00_8034 (CSMR4)
0xFC00_8040 (CSMR5)

Access: User read/write

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

Figure 18-2. Chip-Select Mask Registers (CSMR*n*)

Table 18-4. 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 FB_CS*n* is 2^n; n = (number of bits set in respective CSMR[BAM]) + 16. For example, if CSAR0 equals 0x0000 and CSMR0[BAM] equals 0x0008, FB_CS0 addresses two discontinuous 64 KB memory blocks: one from 0x40_0000 – 0x40_FFFF and one from 0x48_0000 – 0x48_FFFF. Likewise, for FB_CS0 to access 32 MB of address space starting at location 0x00_0000, FB_CS1 must begin at the next byte after FB_CS0 for a 16 MB address space. Therefore, CSAR0 equals 0x0000, CSMR0[BAM] equals 0x01FF, CSAR1 equals 0x0200, and CSMR1[BAM] equals 0x00FF.
15–9	Reserved, must be cleared.
8 WP	Write protect. Controls write accesses to the address range in the corresponding CSAR. Attempting to write to the range of addresses for which CSAR*n*[WP] is set results in a bus error termination of the internal cycle and no external cycle. 0 Read and write accesses are allowed 1 Only read accesses are allowed
7–1	Reserved, must be cleared.
0 V	Valid bit. Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. Programmed chip-selects do not assert until V bit is set (except for FB_CS0, which acts as the global chip-select). Reset clears each CSMR*n*[V]. **Note:** At reset, no chip-select other than FB_CS0 can be used until the CSMR0[V] is set. Afterward, FB_CS[5:0] functions as programmed. 0 Chip-select invalid 1 Chip-select valid

18.3.3 Chip-Select Control Registers (CSCR0 – CSCR5)

Each CSCR*n* controls the auto-acknowledge, address setup and hold times, port size, burst capability, and number of wait states. To support the global chip-select, FB_CS0, the CSCR0 reset values differ from the other CSCRs. 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	See Note	1	See Note	See Note	1	0	0	0	0	0
Reset: CSCR1–5	0	0	0	0	0	0	See Note	1	0	0	1	0	0	0	0	0

Note: The SBM reset value is determined by the chosen chip configuration. See SBM field description in Table 18-5 for more information.

Note: The PS reset value depends upon the chosen chip configuration.

Figure 18-3. Chip-Select Control Registers (CSCR*n*)

Table 18-5. 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 FB_TS asserts. 00 Assert FB_CS*n* on first rising clock edge after address is asserted. (Default FB_CS*n*) 01 Assert FB_CS*n* on second rising clock edge after address is asserted. 10 Assert FB_CS*n* on third rising clock edge after address is asserted. 11 Assert FB_CS*n* on fourth rising clock edge after address is asserted. (Default FB_CS0)

Table 18-5. CSCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>19–18
RDAH</td><td>Read address hold or deselect. This field controls the address and attribute hold time after the termination during a read cycle that hits in the chip-select address space.
Note: The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.
The number of cycles the address and attributes are held after FB_CS*n* 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
(FB_CS*n* Default)</td><td>1 cycle</td><td>0 cycles</td></tr>
<tr><td>01</td><td>2 cycles</td><td>1 cycles</td></tr>
<tr><td>10</td><td>3 cycles</td><td>2 cycles</td></tr>
<tr><td>11
(FB_CS0 Default)</td><td>4 cycles</td><td>3 cycles</td></tr>
</table>
</td></tr>
<tr><td>17–16
WRAH</td><td>Write address hold or deselect. This field controls the address, data, and attribute hold time after the termination of a write cycle that hits in the chip-select address space.
Note: The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.
00 Hold address and attributes one cycle after FB_CS*n* negates on writes. (Default FB_CS*n*)
01 Hold address and attributes two cycles after FB_CS*n* negates on writes.
10 Hold address and attributes three cycles after FB_CS*n* negates on writes.
11 Hold address and attributes four cycles after FB_CS*n* negates on writes. (Default FB_CS0)</td></tr>
<tr><td>15–10
WS</td><td>Wait states. The number of wait states inserted after FB_CS*n* asserts and before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0x3F inserts 63 wait states). If AA is reserved, FB_TA must be asserted by the external system regardless of the number of generated wait states. In that case, the external transfer acknowledge ends the cycle. An external FB_TA supersedes the generation of an internal FB_TA.</td></tr>
<tr><td>9
SBM</td><td>Split bus mode. For proper operation of the chip select signals, this bit must be set when the SDRAM controller is in DDR mode.
0 Device is not in split bus mode (SDRAM controller is in SDR mode).
1 Device is in split bus mode (SDRAM controller is in DDR mode).
Note: Placing the device in split bus mode is only controlled by BOOTMOD or override signals (,” for more details). This bit is only used to provide correct operation of the chip select signals.</td></tr>
<tr><td>8
AA</td><td>Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address.
0 No internal FB_TA is asserted. Cycle is terminated externally
1 Internal transfer acknowledge is asserted as specified by WS

Note: If AA is set for a corresponding FB_CS*n* and the external system asserts an external FB_TA before the wait-state countdown asserts the internal FB_TA, the cycle is terminated. Burst cycles increment the address bus between each internal termination.</td></tr>
</table>

Table 18-5. CSCR*n* Field Descriptions (continued)

Field	Description
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
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.

18.4 Functional Description

18.4.1 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 18.3.1, “Chip-Select Address Registers (CSAR0 – CSAR5).”
- Chip-select mask registers (CSMR*n*) provide 16-bit address masking and access control. See Section 18.3.2, “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 18.3.3, “Chip-Select Control Registers (CSCR0 – CSCR5).”

$\overline{\text{FB\_CS0}}$ is a global chip-select after reset and provides external boot memory capability.

18.4.1.1 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 18-6.

Table 18-6. 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. If CSMR[WP] is set and a write access is performed, the internal bus cycle terminates with a bus error, no chip select is asserted, and no external bus cycle is performed.
No	The chip-select signals are not driven. However, the FlexBus runs an external bus cycle with external termination.
Yes, multiple CSARs	The chip-select signals are driven. However, they are driven using an external burst-inhibited bus cycle with external termination on a 32-bit port.

18.4.1.2 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. Its data bus must be connected to FB_D[7:0] 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 18-4 for a graphical connection.

18.4.1.3 Global Chip-Select Operation

$\overline{\mathrm{FB\_CS0}}$, 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, $\overline{\mathrm{FB\_CS0}}$ is asserted for every external access. No other chip-select can be used until the valid bit, CSMR0[V], is set; at this point $\overline{\mathrm{FB\_CS0}}$ functions as configured. After this, $\overline{\mathrm{FB\_CS}}$[5:1] can be used as well. At reset, the logic levels on the FB_A20 signals determine global chip-select port size.

See ,” for more information.

18.4.2 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 ($\overline{\mathrm{FB\_TS}}$, $\overline{\mathrm{FB\_TA}}$, $\overline{\mathrm{FB\_CS}n}$, $\overline{\mathrm{FB\_OE}}$, $\overline{\mathrm{FB\_BE/BWE}}$[3:0])
- Attribute signals (FB_R/$\overline{\mathrm{W}}$)

The address, write data, FB_TS, 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 18.3.3, “Chip-Select Control Registers (CSCR0 – CSCR5).”

18.4.3 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 18-4 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] (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.

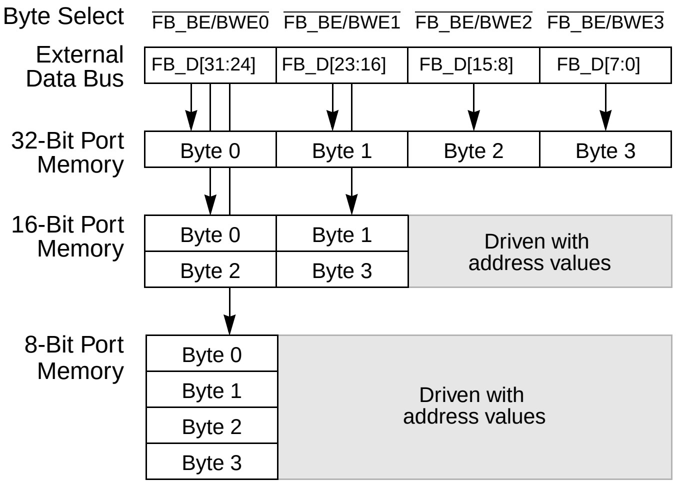


Figure 18-4. Connections for External Memory Port Sizes (CSCR*n*[SBM] = 0)

Figure 18-5 shows the byte lanes that external memory connects to and the sequential transfers of a longword transfer for the supported port sizes when in split bus mode.

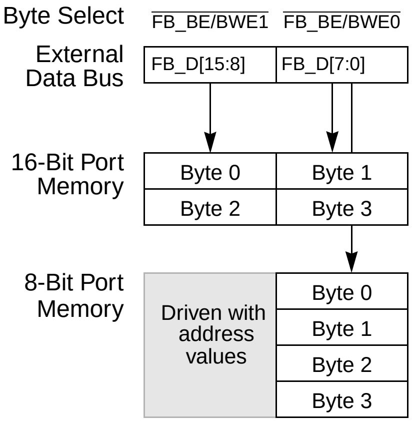


Figure 18-5. Connections for External Memory Port Sizes (CSCR*n*[SBM] = 1)

18.4.4 Bus Cycle Execution

As shown in Figure 18-8 and Figure 18-10, basic bus operations occur in four clocks:

1. S0: At the first clock edge, the address, attributes, and $\overline{\text{FB\_TS}}$ are driven.
2. S1: $\overline{\text{FB\_CS}n}$ is asserted at the second rising clock edge to indicate the device selected; by that time, the address and attributes are valid and stable. $\overline{\text{FB\_TS}}$ 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 $\overline{\text{FB\_CS}n}$ negates. For a read transfer, data is also driven into the device during this cycle.

   External slave asserts $\overline{\text{FB\_TA}}$ at this clock edge.
3. S2: Read data and $\overline{\text{FB\_TA}}$ are sampled on the third clock edge. $\overline{\text{FB\_TA}}$ can be negated after this edge and read data can then be tri-stated.
4. S3: $\overline{\text{FB\_CS}n}$ is negated at the fourth rising clock edge. This last clock of the bus cycle uses what would be an idle clock between cycles to provide hold time for address, attributes, and write data.

18.4.4.1 Data Transfer Cycle States

An on-chip state machine controls the data-transfer operation in the device. Figure 18-6 shows the state-transition diagram for basic read and write cycles.

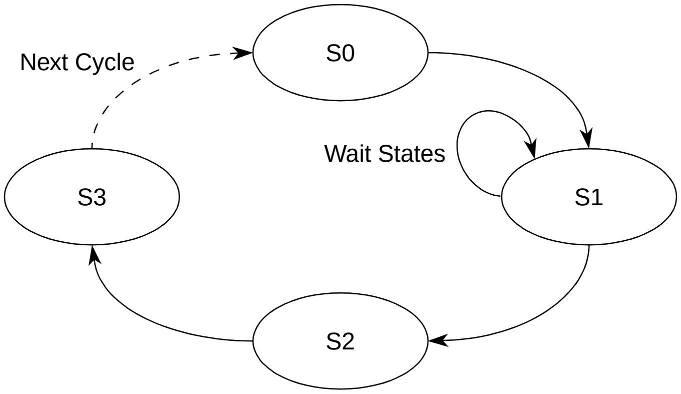


Figure 18-6. Data-Transfer-State-Transition Diagram

Table 18-7 describes the states as they appear in subsequent timing diagrams.

Table 18-7. 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 $\overline{\text{FB\_TS}}$, and drives FB_R/$\overline{\text{W}}$ high for a read and low for a write.
S1	All	$\overline{\text{FB\_TS}}$ is negated on the rising edge of FB_CLK, and $\overline{\text{FB\_CS}n}$ 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 $\overline{\text{FB\_TA}}$ is recognized asserted, then the cycle moves on to S2. If $\overline{\text{FB\_TA}}$ is not asserted internally or externally, then the S1 state continues to repeat.
	Read	Data is driven by the external device before the next rising edge of FB_CLK (the rising edge that begins S2) with $\overline{\text{FB\_TA}}$ asserted.
S2	All	For internal termination, $\overline{\text{FB\_CS}n}$ is negated and the internal system bus transfer is completed. For external termination, the external device should negate $\overline{\text{FB\_TA}}$, and the $\overline{\text{FB\_CS}n}$ chip select negates after the rising edge of FB_CLK at the end of S2.
	Read	The processor latches data on the rising clock edge entering S2. The external device can stop driving data after this edge. However, data can be driven until the end of S3 or any additional address hold cycles.
S3	All	Address, data, and FB_R/$\overline{\text{W}}$ go invalid off the rising edge of FB_CLK at the beginning of S3, terminating the read or write cycle.

18.4.5 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.

18.4.5.1 Basic Read Bus Cycle

During a read cycle, the ColdFire device receives data from memory or a peripheral device. Figure 18-7 is a read cycle flowchart.

NOTE

Throughout this chapter FB_D[31:*X*] indicates a 32-, 16-, or 8-bit wide data bus.

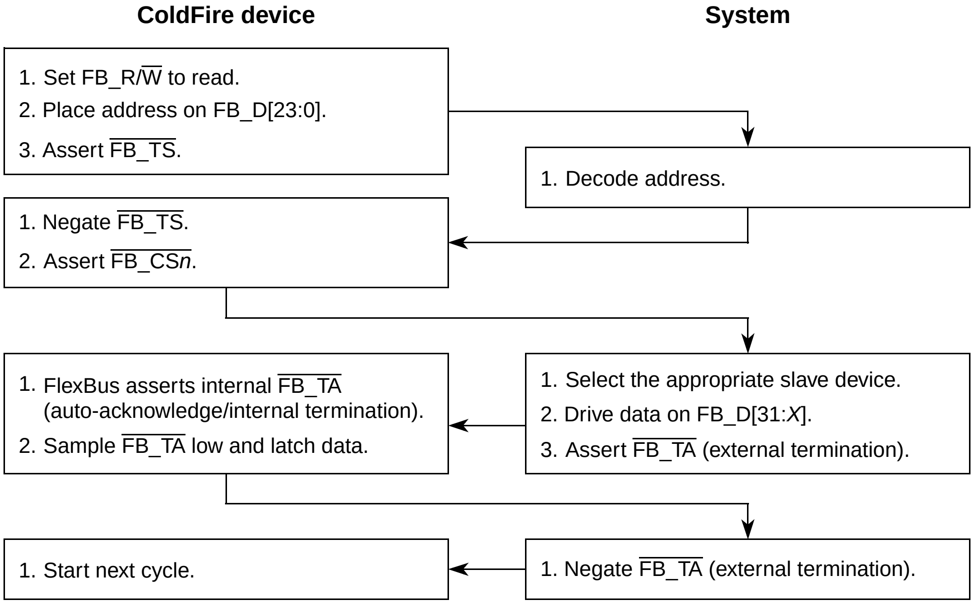

ColdFire device
System
1. Set FB_R/W to read.
2. Place address on FB_D[23:0].
3. Assert FB_TS.
1. Decode address.
1. Negate FB_TS.
2. Assert FB_CS*n*.
1. FlexBus asserts internal FB_TA (auto-acknowledge/internal termination).
2. Sample FB_TA low and latch data.
1. Select the appropriate slave device.
2. Drive data on FB_D[31:*X*].
3. Assert FB_TA (external termination).
1. Start next cycle.
1. Negate FB_TA (external termination).

Figure 18-7. Read Cycle Flowchart

The read cycle timing diagram is shown in Figure 18-8.

NOTE

In the next set of timing diagrams, the dotted lines indicate FB_TA, FB_OE, and FB_CS*n* timing when internal termination is used (CSCR[AA] = 1). The external and internal FB_TA assert at the same time; however, FB_TA is not driven externally for internally-terminated bus cycles.

NOTE

The processor drives the data lines during the first clock cycle of the transfer with the full 32-bit address. This may be ignored by standard connected devices using non-multiplexed address and data buses. However, some applications may find this 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.

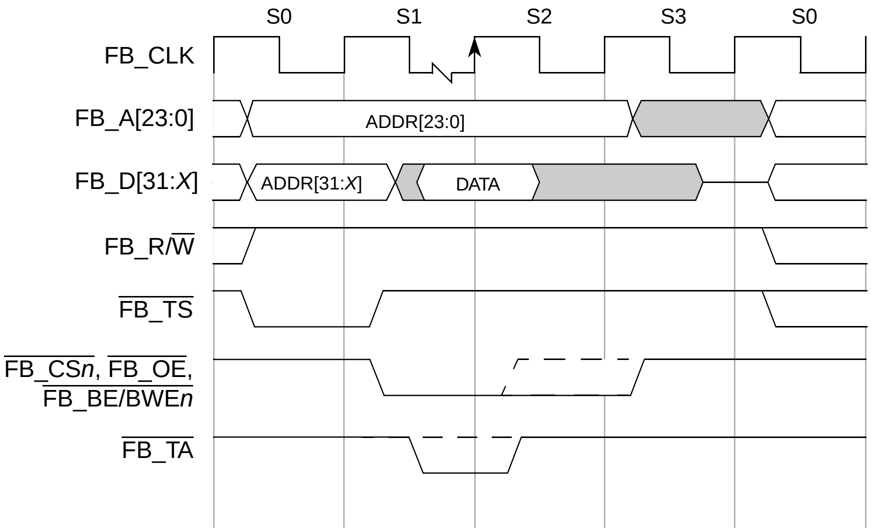


Figure 18-8. Basic Read-Bus Cycle

18.4.5.2 Basic Write Bus Cycle

During a write cycle, the device sends data to memory or to a peripheral device. Figure 18-9 shows the write cycle flowchart.

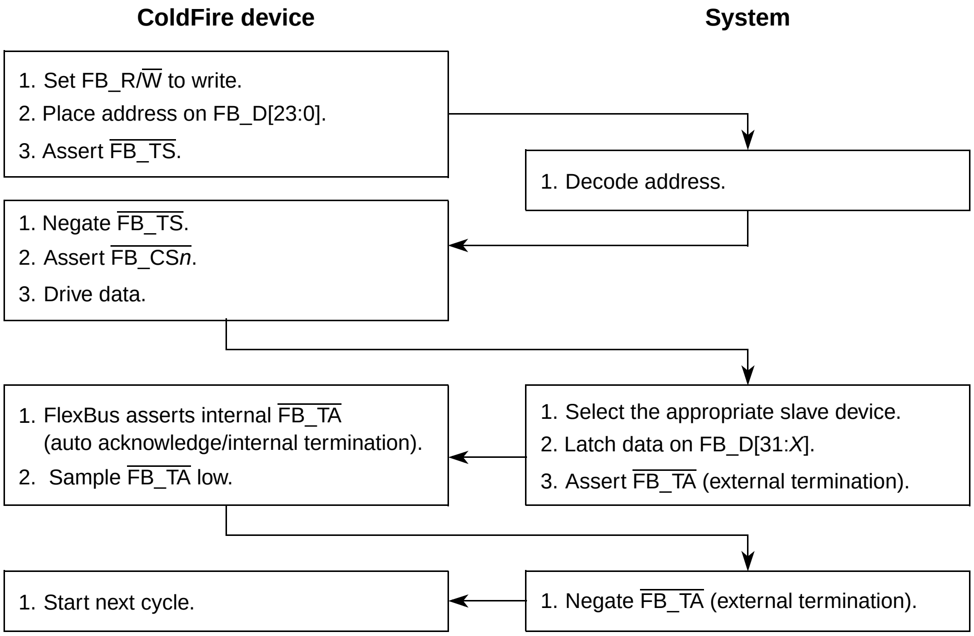

ColdFire device
System
1. Set FB_R/W to write.
2. Place address on FB_D[23:0].
3. Assert FB_TS.
1. Decode address.
1. Negate FB_TS.
2. Assert FB_CSn.
3. Drive data.
1. FlexBus asserts internal FB_TA (auto acknowledge/internal termination).
2. Sample FB_TA low.
1. Select the appropriate slave device.
2. Latch data on FB_D[31:X].
3. Assert FB_TA (external termination).
1. Start next cycle.
1. Negate FB_TA (external termination).

Figure 18-9. Write-Cycle Flowchart

Figure 18-10 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.

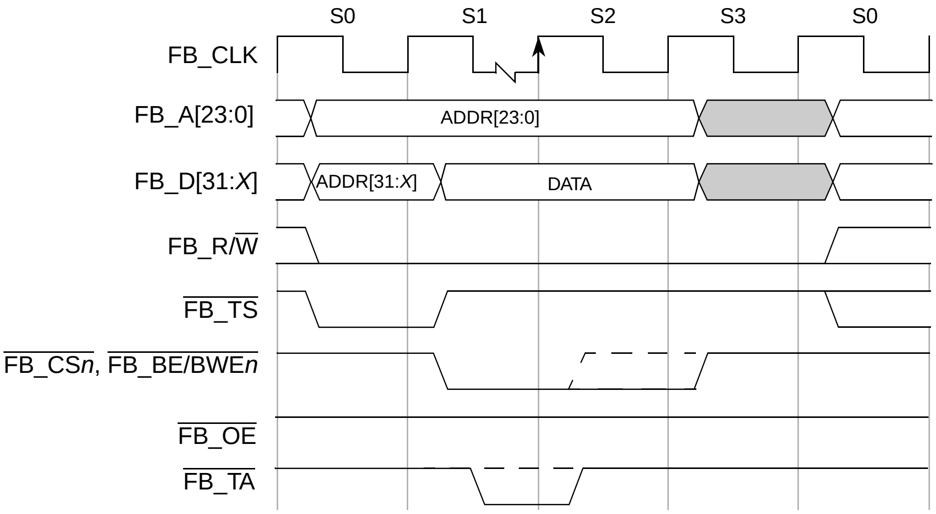


Figure 18-10. Basic Write-Bus Cycle

18.4.5.3 Bus Cycle Sizing

This section shows timing diagrams for various port size scenarios. Figure 18-11 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.

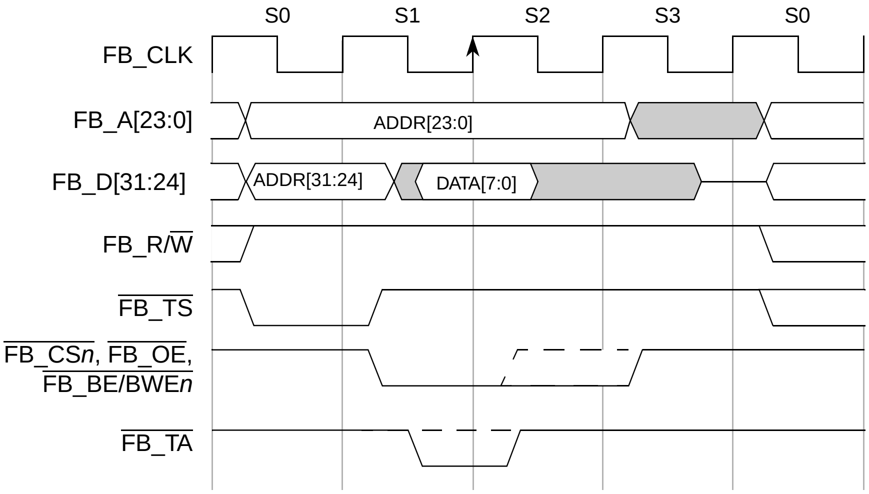


Figure 18-11. Single Byte-Read Transfer

Figure 18-12 shows the similar configuration for a write transfer. The data is driven from the second clock on FB_D[31:24].

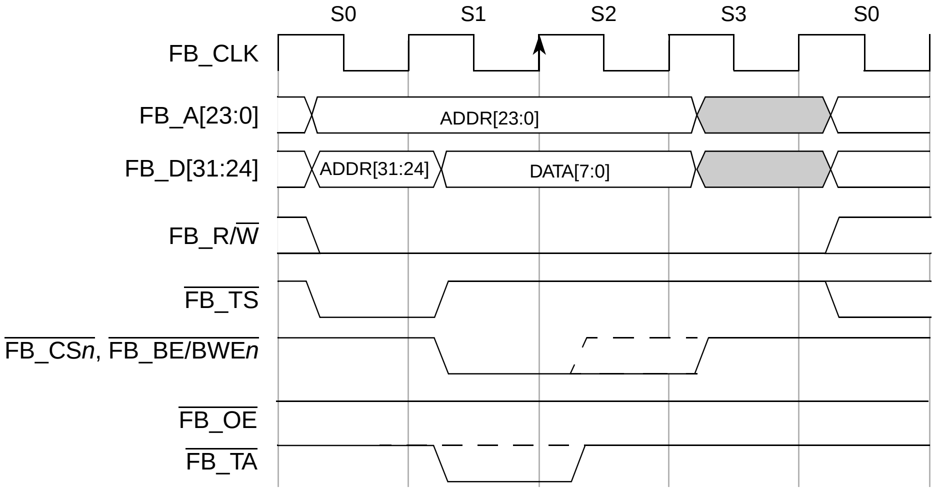


Figure 18-12. Single Byte-Write Transfer

Figure 18-13 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.

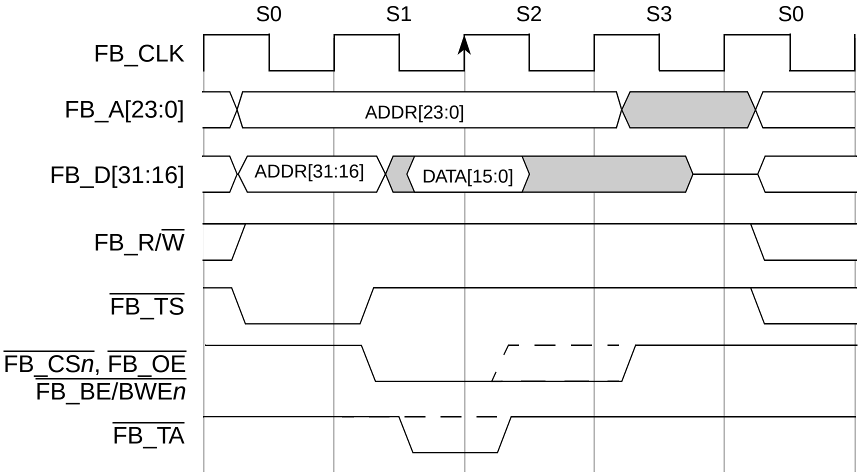


Figure 18-13. Single Word-Read Transfer

Figure 18-14 shows the similar configuration for a write transfer. The data is driven from the second clock on FB_D[31:16].

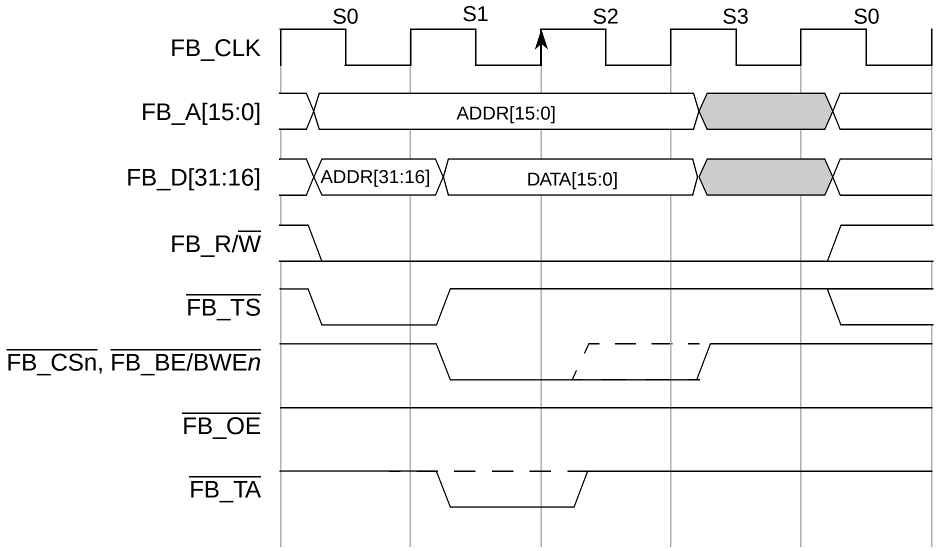


Figure 18-14. Single Word-Write Transfer

Figure 18-15 depicts a longword read from a 32-bit device.

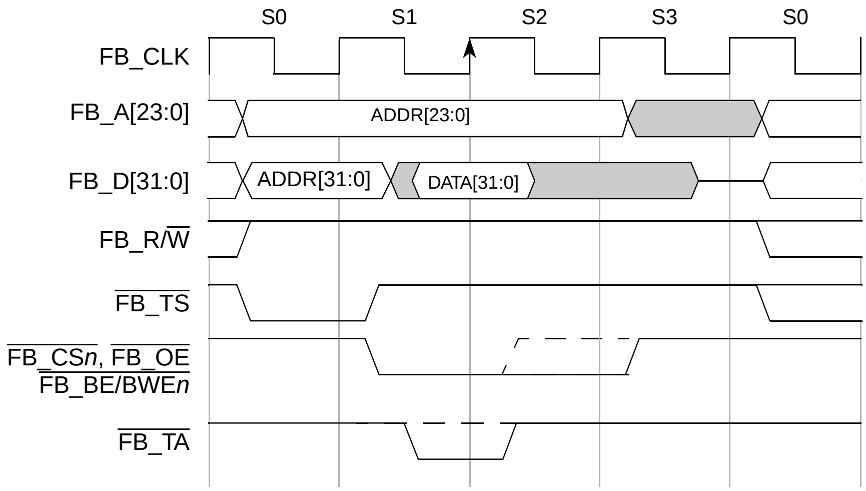


Figure 18-15. Longword-Read Transfer

Figure 18-16 illustrates the longword write to a 32-bit device.

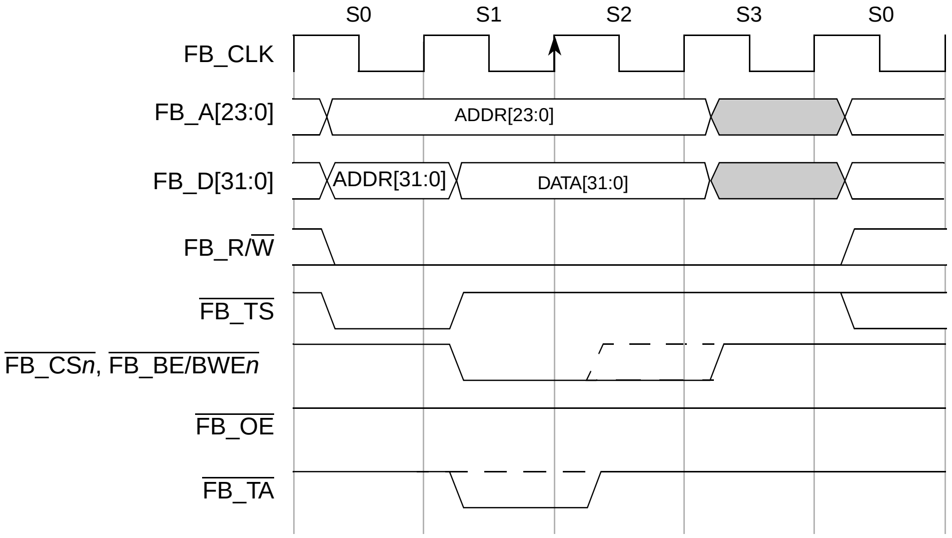


Figure 18-16. Longword-Write Transfer

18.4.5.4 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.

18.4.5.4.1 Wait States

Wait states can be inserted before each beat of a transfer by programming the CSCR*n* registers. Wait states can give the peripheral or memory more time to return read data or sample write data.

Figure 18-17 and Figure 18-18 show the basic read and write bus cycles (also shown in Figure 18-8 and Figure 18-13) with the default of no wait states.

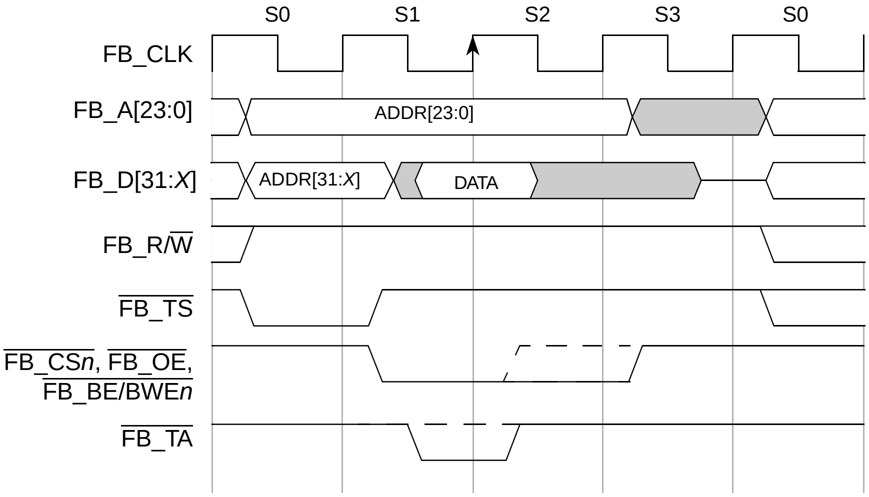


Figure 18-17. Basic Read-Bus Cycle (No Wait States)

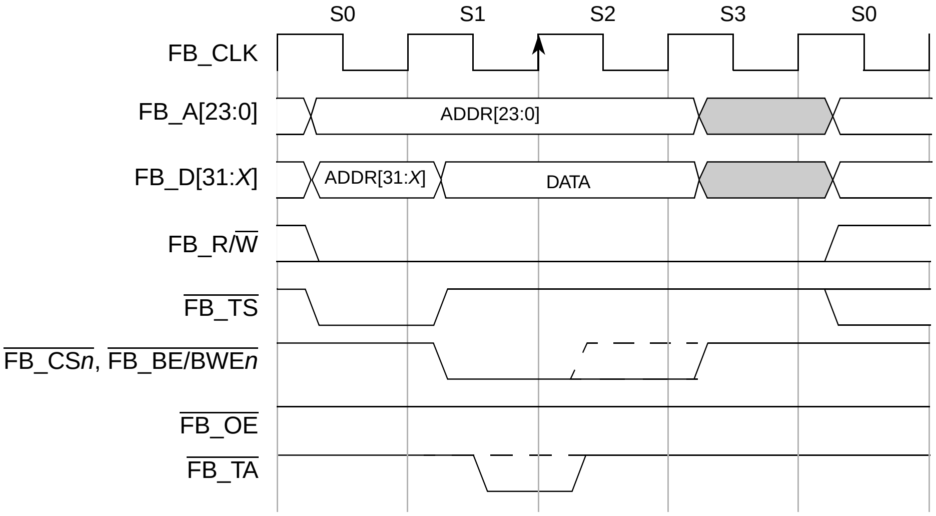


Figure 18-18. Basic Write-Bus Cycle (No Wait States)

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 FB_TA is recognized as asserted. Figure 18-19 and Figure 18-20 show a read and write cycle with one wait state.

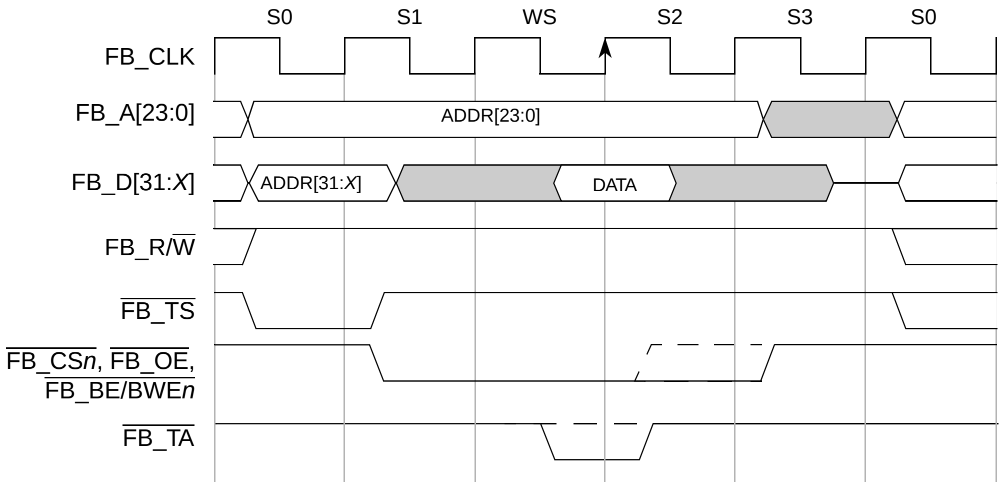


Figure 18-19. Read-Bus Cycle (One Wait State)

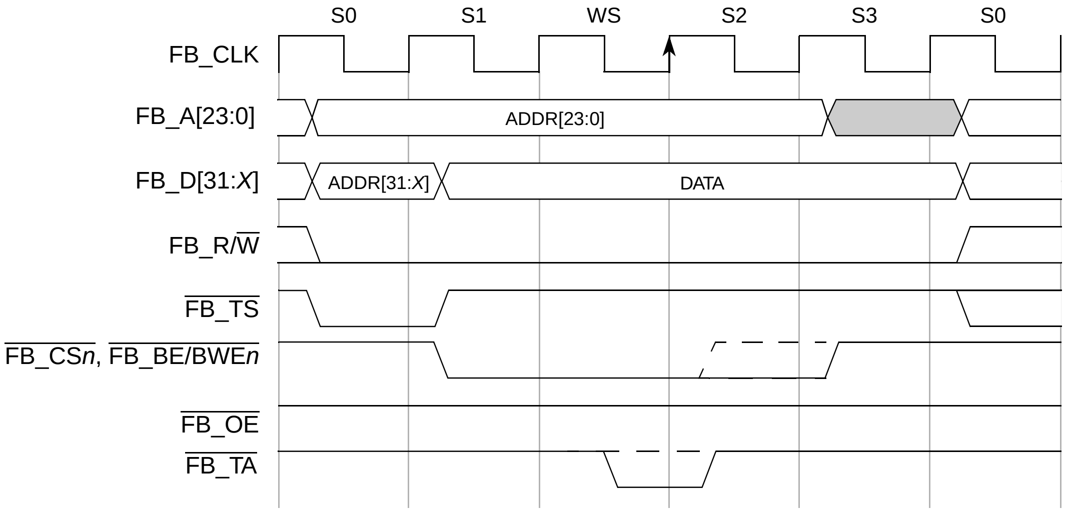


Figure 18-20. Write-Bus Cycle (One Wait State)

18.4.5.4.2 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 (FB_TS) is asserted. Figure 18-21 and Figure 18-22 show read- and write-bus cycles with two clocks of address setup.

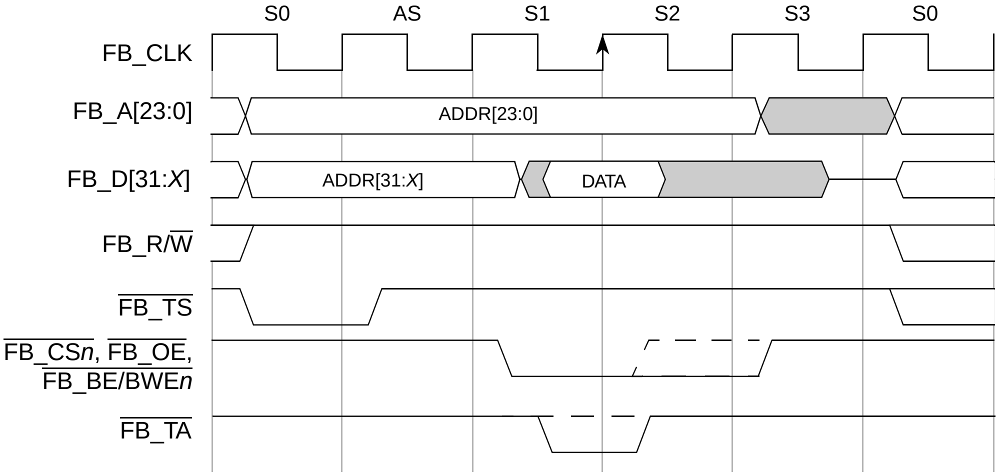


Figure 18-21. Read-Bus Cycle with Two-Clock Address Setup (No Wait States)

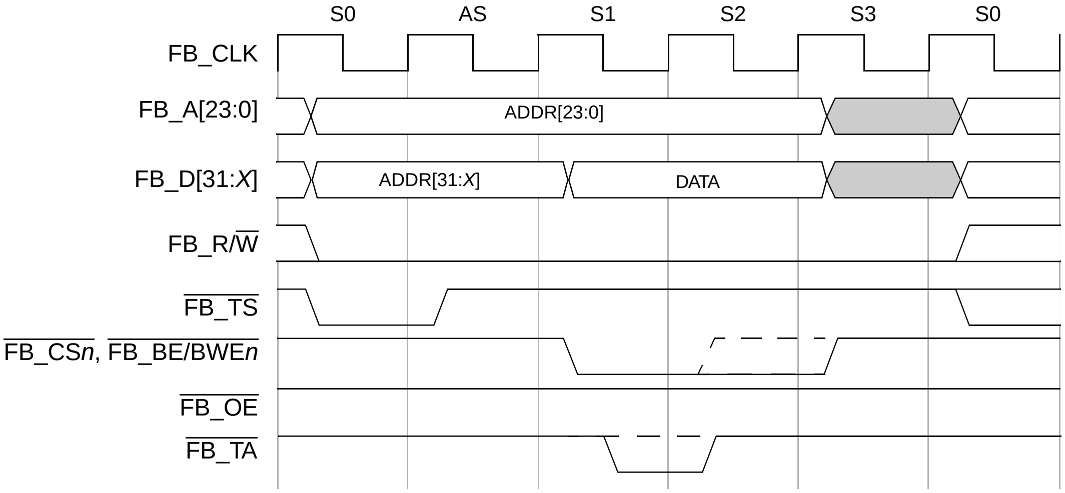


Figure 18-22. Write-Bus Cycle with Two Clock Address Setup (No Wait States)

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 18-23 and Figure 18-24 show read and write bus cycles with two clocks of address hold.

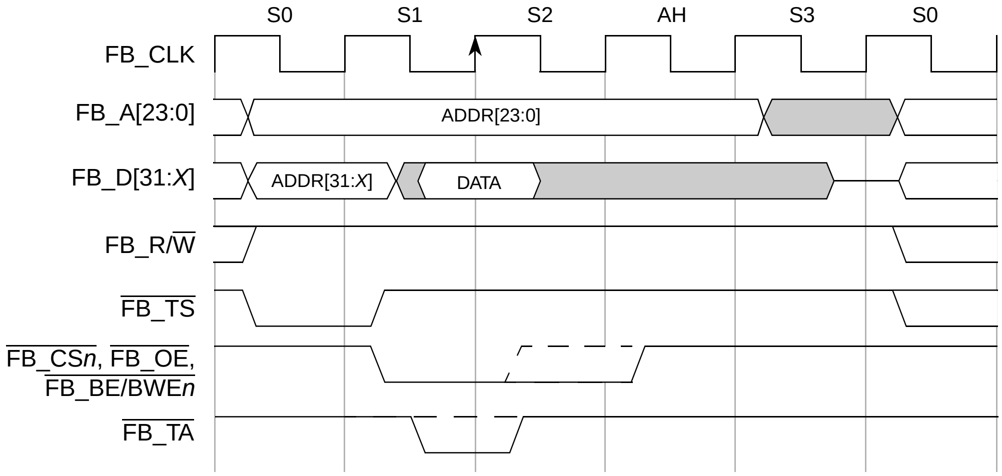


Figure 18-23. Read Cycle with Two-Clock Address Hold (No Wait States)

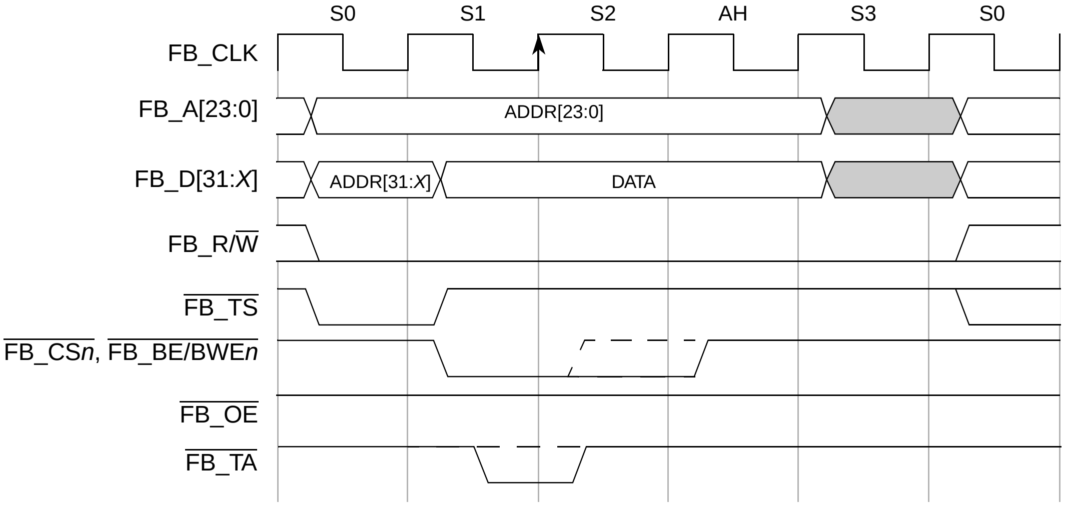


Figure 18-24. Write Cycle with Two-Clock Address Hold (No Wait States)

Figure 18-25 shows a bus cycle using address setup, wait states, and address hold.

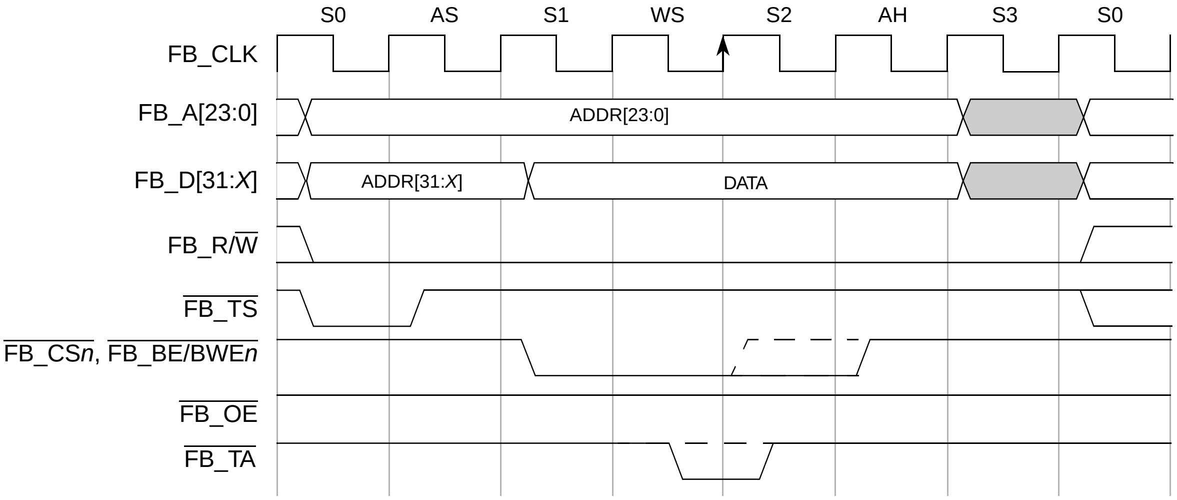


Figure 18-25. Write Cycle with Two-Clock Address Setup and Two-Clock Hold (One Wait State)

18.4.6 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 18-8 shows the result of such transfer translations.

Table 18-8. 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
1*x* (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 CSCR*n* registers enable bursting for reads, writes, or both. Memory spaces can be declared burst-inhibited for reads and writes by clearing the appropriate CSCR*n*[BSTR,BSTW] bits.

Figure 18-26 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

In non-multiplexed address/data mode, the address on FB_A increments only during internally-terminated burst cycles. The first address is driven throughout the entire burst for externally-terminated cycles.

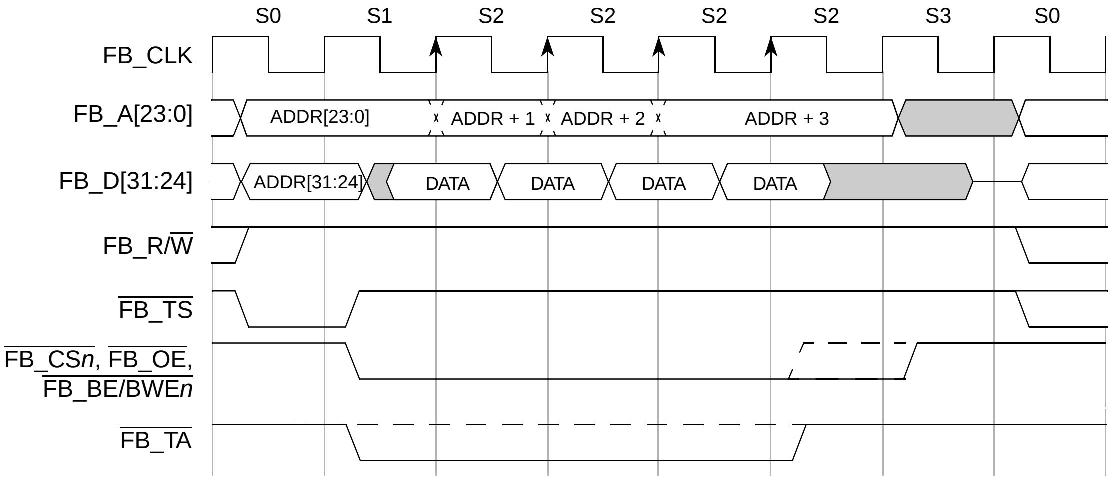


Figure 18-26. Longword-Read Burst from 8-Bit Port 2-1-1-1 (No Wait States)

Figure 18-27 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 (CSCR*n*[WS] = 0), one wait state is added. Otherwise, the programmed number of wait states are used.

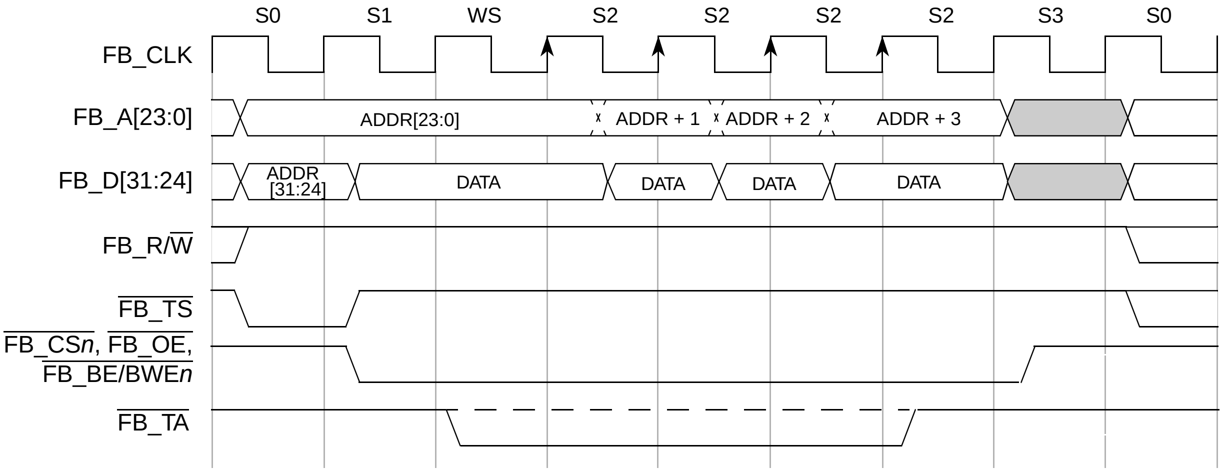


Figure 18-27. Longword-Write Burst to 8-Bit Port 3-1-1-1 (No Wait States)

Figure 18-28 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.

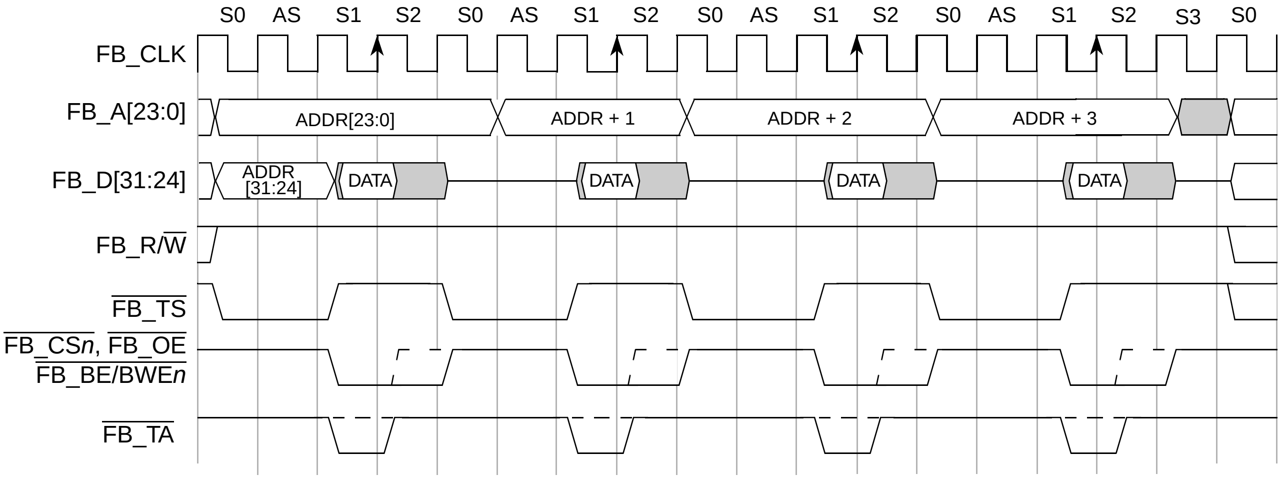


Figure 18-28. Longword-Read Burst-Inhibited from 8-Bit Port (No Wait States)

Figure 18-29 shows a longword write to an 8-bit device with burst inhibited. The transfer results in four individual transfers.

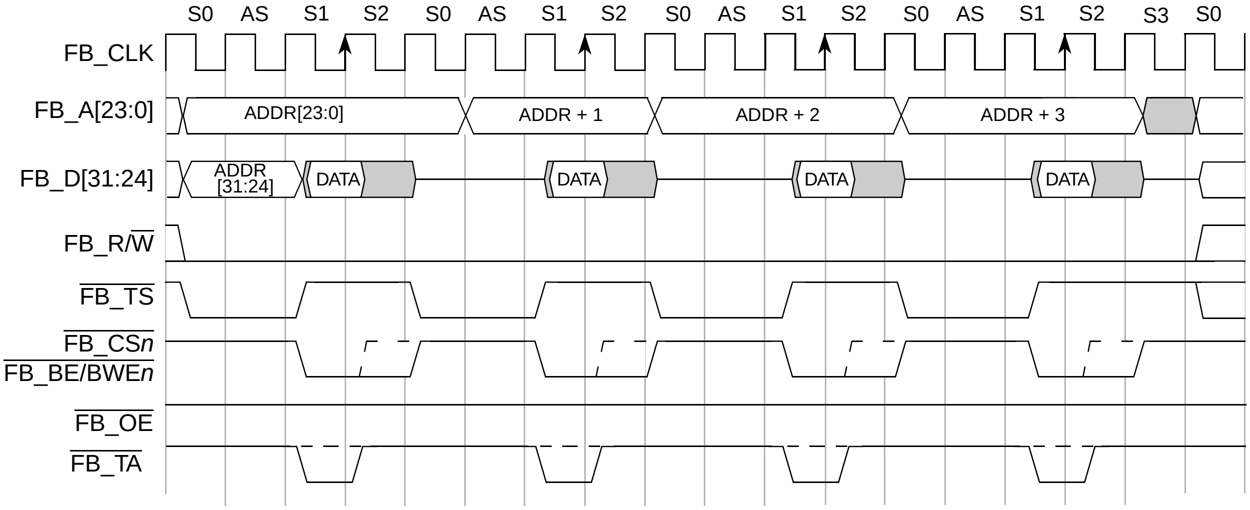


Figure 18-29. Longword-Write Burst-Inhibited to 8-Bit Port (No Wait States)

Figure 18-30 illustrates another read burst transfer, but in this case a wait state is added between individual beats.

NOTE

CSCR*n*[WS] determines the number of wait states in the first beat. However, for subsequent beats, the CSCR*n*[WS] (or CSCR*n*[SWS] if CSCR*n*[SWSEN] is set) determines the number of wait states.

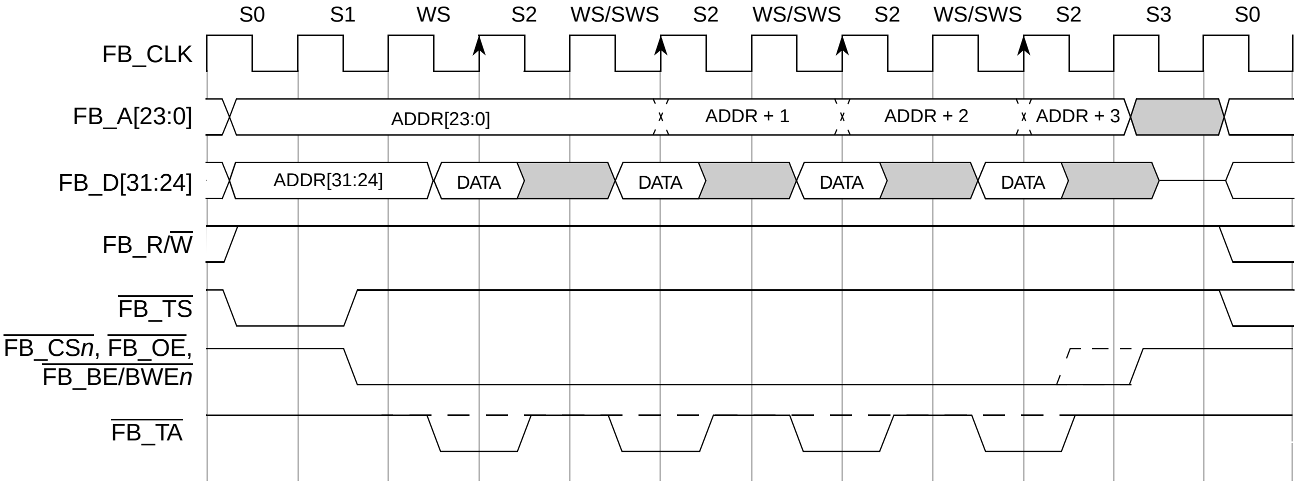


Figure 18-30. Longword-Read Burst from 8-Bit Port 3-2-2-2 (One Wait State)

Figure 18-30 illustrates a write burst transfer with one wait state.

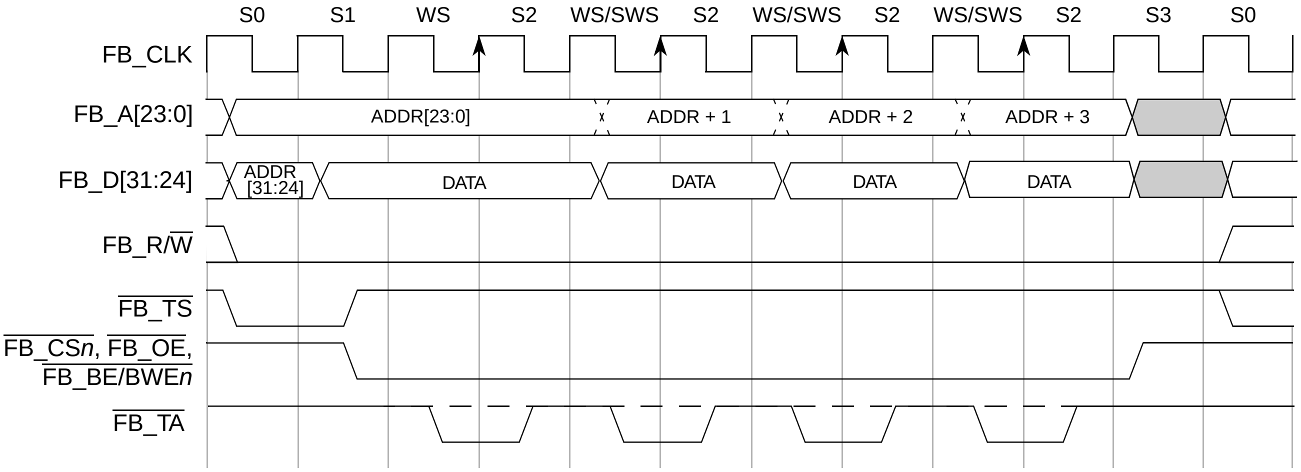


Figure 18-31. Longword-Write Burst to 8-Bit Port 3-2-2-2 (One Wait State)

If address setup and hold are used, only the first and last beat of the burst cycle are affected. Figure 18-32 shows a read cycle with one clock of address setup and address hold.

NOTE

In non-multiplexed address/data mode, the address on FB_A increments only during internally-terminated burst cycles (CSCR*n*[AA] = 1). The attached device must be able to account for this, or a wait state must be added. The first address is driven throughout the entire burst for externally-terminated cycles.

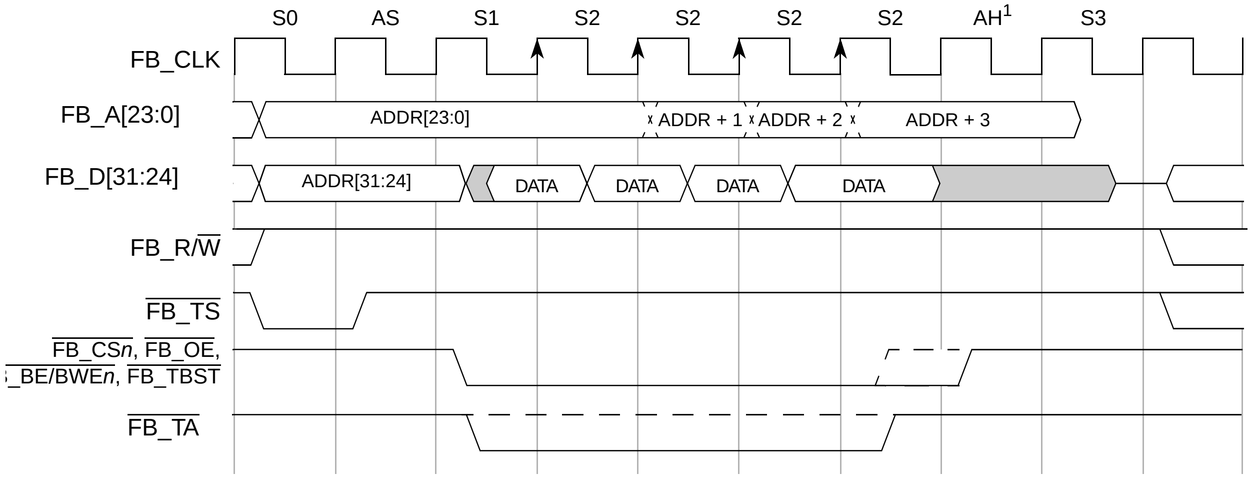


Figure 18-32. Longword-Read Burst from 8-Bit Port 3-1-1-1 (Address Setup and Hold)

Figure 18-33 shows a write cycle with one clock of address setup and address hold.

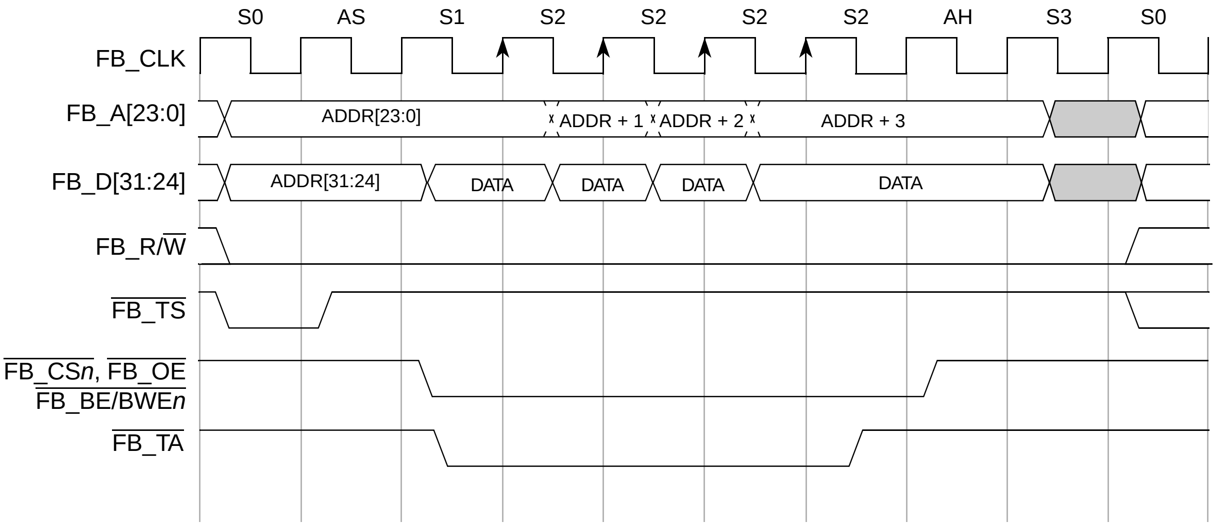


Figure 18-33. Longword-Write Burst to 8-Bit Port 3-1-1-1 (Address Setup and Hold)

18.4.7 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 18-1 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 18-1. 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 18-2 differs from the one in Example 18-1 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 18-2. A Misaligned Word Transfer (32-Bit Port)

18.4.8 Bus Errors

If the auto-acknowledge feature is not enabled for the address that generates the error, the bus cycle can be terminated by asserting $\overline{\text{FB\_TA}}$. If the processor must manage a bus error differently, asserting an interrupt to the core along with $\overline{\text{FB\_TA}}$ when the bus error occurs can invoke an interrupt handler.

The device also includes a bus monitor that generates a bus error for unterminated cycles.

Chapter 19
SDRAM Controller (SDRAMC)

19.1 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, the data bus is switched dynamically between the SDRAM controller and the FlexBus module. However, when the SDRAM controller is in DDR mode, D[31:16] is dedicated to the SDRAM data bus and D[15:0] is dedicated to the FlexBus data bus.

19.1.1 Block Diagram

Block diagram of the SDRAM controller:

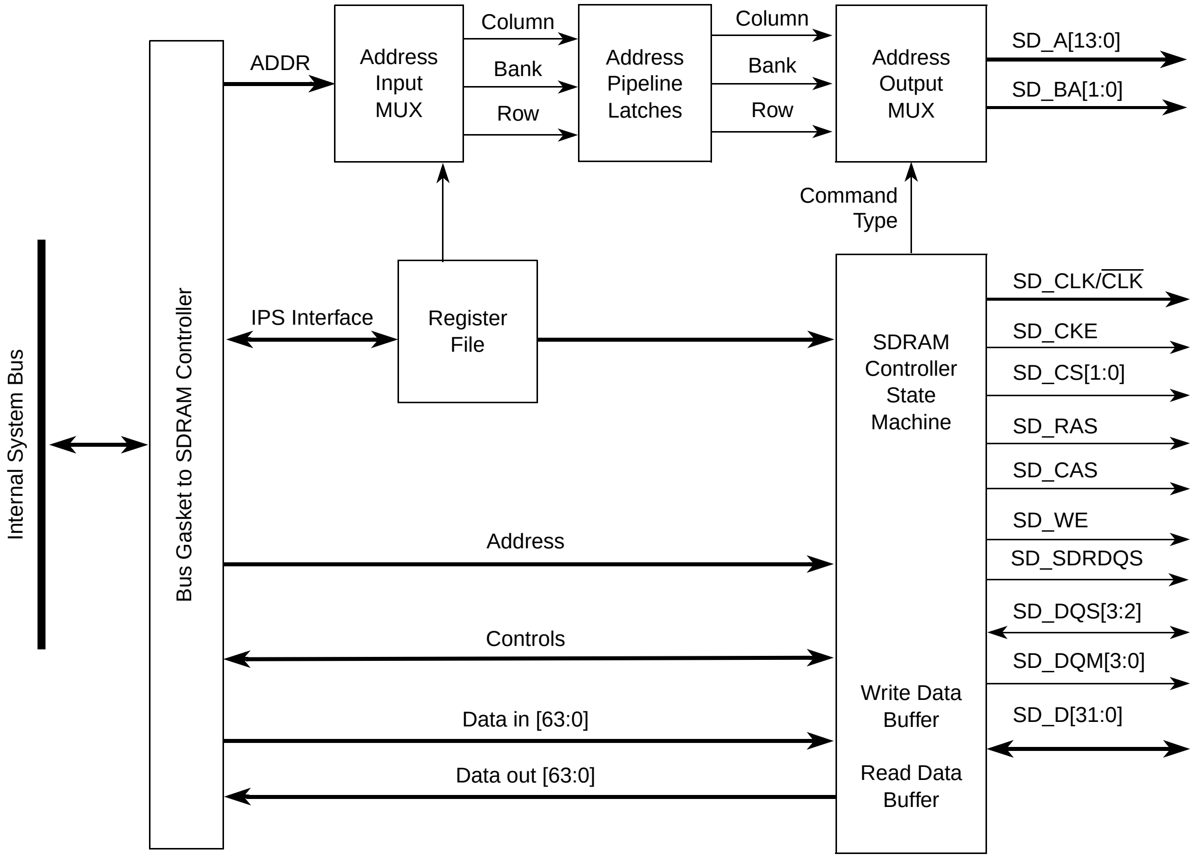


Figure 19-1. SDRAM Controller Block Diagram

19.1.2 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

19.1.3 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.

19.2 External Signal Description

This section introduces the signal names used in this chapter.

Table 19-1. 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 19-1. 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
$\overline{\text{SD\_CAS}}$	O	Column address strobe/command input. Along with $\overline{\text{SD\_CS}}$, $\overline{\text{SD\_RAS}}$, and $\overline{\text{SD\_WE}}$, defines the current command.	
		State Meaning	See Table 19-12 for the SDRAM commands.
		Timing	Assertion/Negation — Occurs synchronously with SD_CLK
$\overline{\text{SD\_RAS}}$	O	Row address strobe/command input. Along with $\overline{\text{SD\_CS}}$, $\overline{\text{SD\_CAS}}$, and $\overline{\text{SD\_WE}}$, defines the current command.	
		State Meaning	See Table 19-12 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, $\overline{\text{SD\_CLK}}$, and SD_CKE, are disabled during self-refresh.	
		State Meaning	Asserted — Activates internal clock signals and device input buffers and output drivers. Negated —Deactivates internal clock signals and device input buffers and output drivers.
		Timing	Assertion — Asynchronous for self-refresh exit and for output disable Negation — Occurs synchronously with SD_CLK
SD_CLK $\overline{\text{SD\_CLK}}$	O	SD_CLK and $\overline{\text{SD\_CLK}}$ are differential clock outputs. All address and control output signals are sent on the crossing of the positive edge of SD_CLK and the negative edge of $\overline{\text{SD\_CLK}}$. Output data is referenced to the crossing of SD_CLK and $\overline{\text{SD\_CLK}}$ (both directions of crossing).	
		Timing	Command signals occur synchronously with the rising edge of this clock. Data signals can change on the rising and falling edge of the clock.
$\overline{\text{SD\_CS[1:0]}}$	O	$\overline{\text{SD\_CS}}$ provides external bank selection on systems with multiple banks. $\overline{\text{SD\_CS}}$ is considered part of the command code.	
		State Meaning	Asserted — Commands for the selected chip occur Negated — All commands are masked.
		Timing	Assertion/Negation — Occurs synchronously with SD_CLK
SD_DATA[31:0]	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 $\overline{\text{SD\_CLK}}$. High Impedance - Depending on the OE_RULE bit in SDCFG1, the SD_DATA bus can be in high impedance until a write occurs or only when a read occurs.

Table 19-1. 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 $\overline{\text{SD\_CLK}}$.
$\overline{\text{SD\_WE}}$	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 19-12 for SDRAM commands.
		Timing	Assertion/Negation— Occurs synchronously with SD_CLK.

19.3 Interface Recommendations

19.3.1 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 19-5 and Table 19-6 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 19-2. 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 19-3. 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			
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			

Table 19-3. 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
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			
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 19-4. 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 19-5. 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			
128 Mbits	8M x 16 bit	12 x 9 x 4	00	—	—	—	—	RA11-0	BA1-0	CA8-0
	16M x 8 bit	12 x 10 x 4	00	—	—	—	CA9			
		13 x 9 x 4	01	—	—	—	RA12			
	32M x 4 bit	12 x 11 x 4	00	—	—	CA11	CA9			
		13 x 10 x 4	01	—	—	CA9	RA12			
		14 x 9 x 4	10	—	—	RA13	RA12			
256 Mbits	16M x 16 bit	12 x 10 x 4	00	—	—	—	CA9	RA11-0	BA1-0	CA8-0
		13 x 9 x 4	01	—	—	—	RA12			
	32M x 8 bit	12 x 11 x 4	00	—	—	CA11	CA9			
		13 x 10 x 4	01	—	—	CA9	RA12			
		14 x 9 x 4	10	—	—	RA13	RA12			
	64M x 4 bit	12 x 12 x 4	00	—	CA12	CA11	CA9			
		13 x 11 x 4	01	—	CA11	CA9	RA12			
		14 x 10 x 4	10	—	CA9	RA13	RA12			

Table 19-5. 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
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.

19.3.2 SDRAM SDR Connections

Figure 19-2 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.

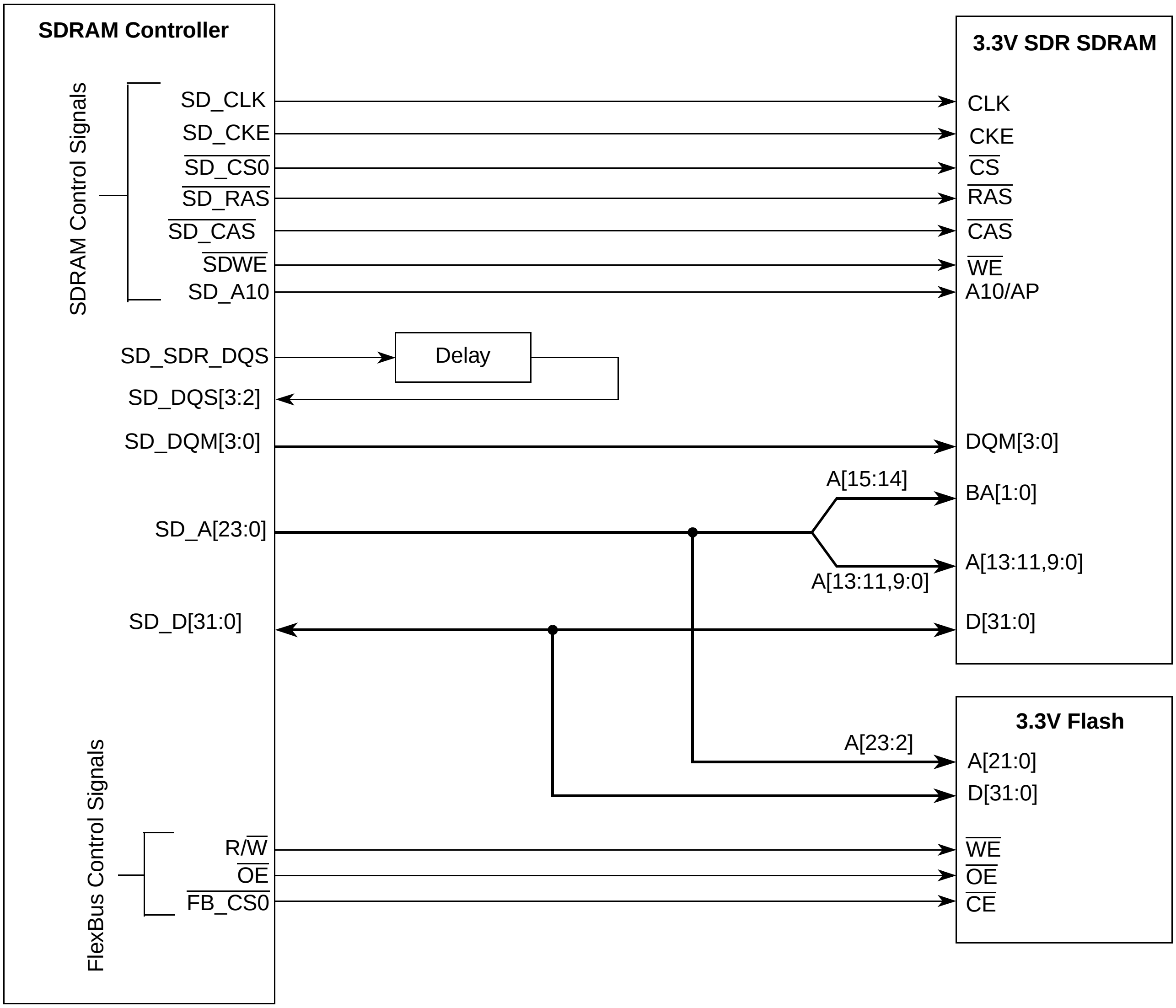


Figure 19-2. Example 3.3V, 32-bit SDR SDRAM System

19.3.3 SDRAM DDR Component Connections

Figure 19-3 shows a block diagram using 16-bit wide DDR SDRAM (such as Micron MT46V8M16) and flash (such as Spansion AM29DBB160G).

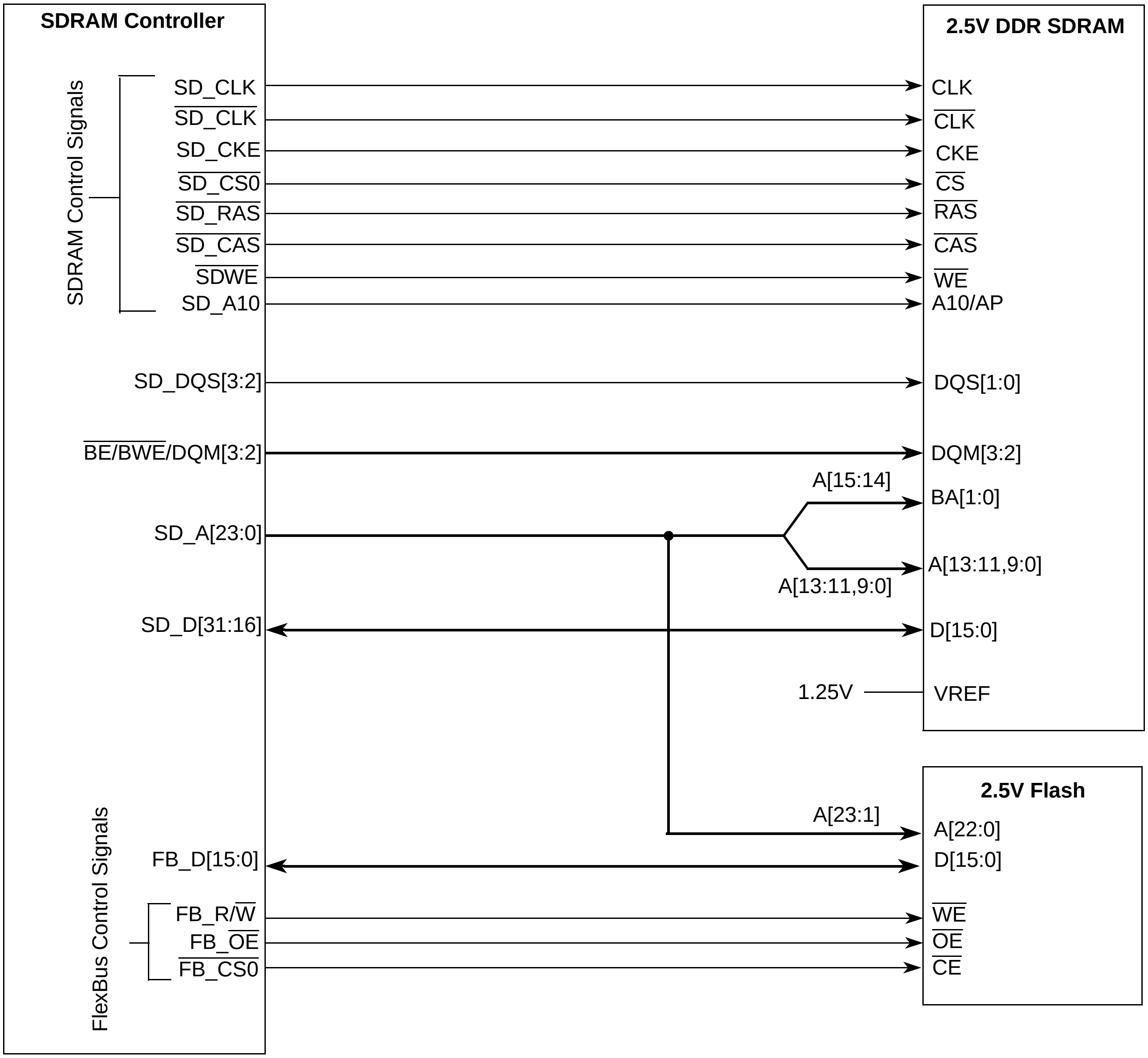


Figure 19-3. Example 2.5V, 16-bit DDR SDRAM System

19.3.4 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 $\overline{\mathrm{SD\_CLK}}$ 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.

19.3.4.1 Termination Example

Figure 19-4 shows the recommended termination circuitry for DDR SDRAM signals.

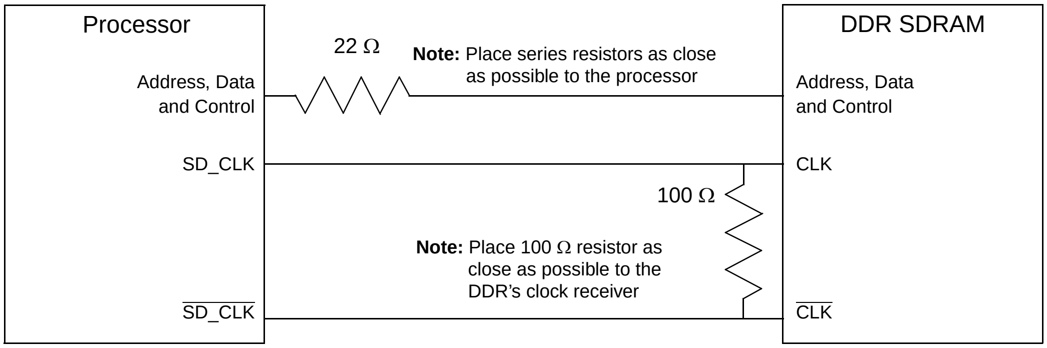

Processor
Address, Data and Control
SD_CLK
$\overline{\mathrm{SD\_CLK}}$
22 Ω
Note: Place series resistors as close as possible to the processor
100 Ω
Note: Place 100 Ω resistor as close as possible to the DDR's clock receiver
DDR SDRAM
Address, Data and Control
CLK
$\overline{\mathrm{CLK}}$

Figure 19-4. DDR SDRAM Termination Circuit

19.4 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

Table 19-6 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 19-6. 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	19.4.1/19-14
0xFC0B_8004	SDRAM Control Register (SDCR)	32	R/W	0x0000_0000	19.4.2/19-15
0xFC0B_8008	SDRAM Configuration Register 1 (SDCFG1)	32	R/W	0x0000_0000	19.4.3/19-17
0xFC0B_800C	SDRAM Configuration Register 2 (SDCFG2)	32	R/W	0x0000_0000	19.4.4/19-19
0xFC0B_8110	SDRAM Chip Select 0 Configuration (SDCS0)	32	R/W	0x0000_0000	19.4.5/19-20
0xFC0B_8114	SDRAM Chip Select 1 Configuration (SDCS1)	32	R/W	0x0000_0000	19.4.5/19-20

19.4.1 SDRAM Mode/Extended Mode Register (SDMR)

The SDMR (Figure 19-5) writes to the mode and extended mode registers physically residing within the SDRAM chips. These registers must be programmed during SDRAM initialization. See Section 19.6, “Initialization/Application Information” for more information on the initialization sequence.

Address: 0xFC0B_8000 (SDMR)

Access: SDCR[MODE_EN] = 0 User read-only
SDCR[MODE_EN] = 1 User read/write

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

Figure 19-5. SDRAM-Mode/Extended-Mode Register (SDMR)

Table 19-7. 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.

19.4.2 SDRAM Control Register (SDCR)

The SDCR (Figure 19-6) controls SDRAMC operating modes, including refresh count and address line muxing.

Address: 0xFC0B_8004 (SDCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MODE_EN	CKE	DDR_MODE	REF_EN	0	0	ADDR_MUX		0	OE_RULE	REF_CNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	MEM_PS	0	DQS_OE		0	0	0	0	0	0	0	0	0	0
W														IREF	IPALL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-6. SDRAM Control Register (SDCR)

Table 19-8. 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 19-8. 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 19-4, and Table 19-5.
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 × SD_CLK period. REF_CNT = (t_{REFI}/ (t_{CK} × 64)) - 1, rounded down to the next integer value. If the SDRAM data sheet does not define t_{REFI}, it can be calculated by t_{REFI} = t_{REF} / #rows.
15–14	Reserved, must be cleared.
13 MEM_PS	Memory data port size. 0 32-bit data bus 1 16-bit data bus
12	Reserved, must be cleared.
11–10 DQS_OE	DQS output enable. Each bit of the DQS_OE field is a master enable for the corresponding SD_DQS*n* signal. DQS_OE[1] (SDCR[11]) enables SD_DQS3 and DQS_OE[0] (SDCR[10]) enables SD_DQS2. 0 SD_DQS*n* can never drive. Use this value in 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.

Table 19-8. SDCR Field Descriptions (continued)

Field	Description
2 IREF	Initiate refresh command. Used to force a software-initiated refresh command. This bit is write-only, reads return zero. 0 Do not generate a refresh command. 1 Generate a refresh command. All $\overline{\text{SD\_CS}n}$ signals are asserted simultaneously. SDCR[CKE] must be set before attempting to generate a software refresh command. **Note:** A software requested refresh is completely independent of the periodic refresh interval counter. Software refresh is only possible when MODE_EN is set.
1 IPALL	Initiate precharge all command. Used to force a software-initiated precharge all command. This bit is write-only, reads return zero. 0 Do not generate a precharge command. 1 Generate a precharge all command. All $\overline{\text{SD\_CS}n}$ signals are asserted simultaneously. SDCR[CKE] must be set before generating a software precharge command. **Note:** Software precharge is only possible when MODE_EN is set. **Note:** Do not set IREF and IPALL at the same time.

19.4.3 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 19-7. SDRAM Configuration Register 1 (SDCFG1)

Table 19-9. 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 × 99 MHz) - 1= 0.98; round to 1. **Note:** Count value is in SD_CLK periods for SDR and DDR modes.

Table 19-9. 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 × 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 × 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.

19.4.4 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 19-8. SDRAM Configuration Register 2 (SDCFG2)

Table 19-10. 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.

19.4.5 SDRAM Chip Select Configuration Registers (SDCS*n*)

These registers define base address and space size of each chip select.

NOTE

Because the SDRAM module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x4000_0000 – 0x7FFF_FFFF. Be sure to set the SDCS*n* registers appropriately.

NOTE

The user should not probe memory on a DDR chip select to determine if memory is connected. If a read is attempted from a DDR SDRAM chip select when there is no memory to respond with the appropriate DQS pulses, the bus cycle hangs. Because no high level bus monitor exists on the device, a reset is the only way to exit the error condition.

Address: 0xFC0B_8110 (SDCS0)
0xFC0B_8114 (SDCS1)

Access: User read/write

<table>
<tr><th></th><th>31</th><th>30</th><th>29</th><th>28</th><th>27</th><th>26</th><th>25</th><th>24</th><th>23</th><th>22</th><th>21</th><th>20</th><th>19</th><th>18</th><th>17</th><th>16</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td colspan="12" rowspan="2">CSBA</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="5" rowspan="2">CSSZ</td></tr>
<tr><td>W</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 19-9. SRAM Chip Select Configuration Register (SDCS*n*)

Table 19-11. SDCS*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–20
CSBA</td><td>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.</td></tr>
<tr><td>19–5</td><td>Reserved, must be cleared.</td></tr>
<tr><td>4–0
CSSZ</td><td>Chip select size.
<table>
<tr><th>CSSZ</th><th>Size</th><th>Address Lines to Compare</th><th>CSSZ</th><th>Size</th><th>Address Lines to Compare</th></tr>
<tr><td>00000</td><td>Disabled</td><td>—</td><td>11000</td><td>32 MByte</td><td>A[31:25]</td></tr>
<tr><td>00001–10001</td><td>Reserved</td><td>Reserved</td><td>11001</td><td>64 MByte</td><td>A[31:26]</td></tr>
<tr><td>10010</td><td>Reserved</td><td>Reserved</td><td>11010</td><td>128 MByte</td><td>A[31:27]</td></tr>
<tr><td>10011</td><td>1 MByte</td><td>A[31:20]</td><td>11011</td><td>256 MByte</td><td>A[31:28]</td></tr>
<tr><td>10100</td><td>2 MByte</td><td>A[31:21]</td><td>11100</td><td>512 MByte</td><td>A[31:29]</td></tr>
<tr><td>10101</td><td>4 MByte</td><td>A[31:22]</td><td>11101</td><td>1 GByte</td><td>A[31:30]</td></tr>
<tr><td>10110</td><td>8 MByte</td><td>A[31:23]</td><td>11110</td><td>2 GByte</td><td>A31</td></tr>
<tr><td>10111</td><td>16 MByte</td><td>A[31:24]</td><td>11111</td><td>4 GByte</td><td>Ignore A[31:20]</td></tr>
</table></td></tr>
</table>

Any chip-select can be enabled or disabled, independent of others. Any chip-select can be allocated any size of address space from 1M to 4G, independent of others. Any chip-select address space can begin at any size-aligned base address, independent of others.

For contiguous memory with different sizes of memory blocks, place largest block at the lowest address and place smaller blocks in descending size order at ascending base addresses.

For example, assume a system with 2 chip selects: CS0=16M, CS1=256M:

CS0CFG = 0x4F000017 = enable 16M @ 0x4F000000-0x4FFFFFFF
CS1CFG = 0x5000001B = enable 256M @ 0x50000000-0x5FFFFFFF

This gives 272 MB total memory, at 0x4F000000-0x5FFFFFFF.

19.5 Functional Description

19.5.1 SDRAM Commands

When an internal bus master accesses SDRAM address space, the memory controller generates the corresponding SDRAM command. Table 19-12 lists SDRAM commands supported by the memory controller.

Table 19-12. SDRAM Commands

Function	Symbol	CKE	$\overline{\text{CS}}$	$\overline{\text{RAS}}$	$\overline{\text{CAS}}$	$\overline{\text{WE}}$	BA[1:0]	A[10]	Other A
Command Inhibit	INH	H	H	X	X	X	X	X	X
No Operation	NOP	H	L	H	H	H	X	X	X
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).

19.5.1.1 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 19.6.4, “Page Management,” for more information.

19.5.1.2 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.

19.5.1.3 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.

19.5.1.4 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.

19.5.1.5 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.

19.5.1.6 Load Mode/Extended Mode Register Command (LMR, LEMR)

All SDRAM devices contain mode registers that configure the timing and burst mode for the SDRAM. These commands access the mode registers that physically reside within the SDRAM devices. During the LMR or LEMR command, SDRAM latches the address and bank buses to load the values into the selected mode register.

NOTE

> The LMR and LEMR commands are only used during SDRAM initialization.

Use the following steps to write the mode register and extended mode register:

1. Set the SDCR[MODE_EN] bit.
2. Write the SDMR[BA] bits to select the mode register.
3. Write the desired mode register value to the SDMR[ADDR]. Do not overwrite the SDMR[BA] values. This step can be performed in the same register write in step 2.
4. Set the SDMR[CMD] bit.
5. For DDR, perform steps 2–4 more than once to write the extended-mode register and the mode register.
6. Clear the SDCR[MODE_EN] bit.

19.5.1.6.1 Mode Register Definition

Figure 19-10 shows a typical mode register definition. This is the SDRAM's mode register, not the SDRAMC's mode/extended mode register (SDMR) defined in Section 19.4.1, "SDRAM Mode/Extended Mode Register (SDMR)." Refer to the SDRAM manufacturer's device data sheet to confirm correct settings.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	0	OP_MODE					CL			BT	BLEN		

Figure 19-10. Typical Mode Register

Table 19-13. 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

Table 19-13. Mode Register Field Descriptions (continued)

Field	Description
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.
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

19.5.1.6.2 DDR Extended Mode Register Definition

Figure 19-12 shows a typical extended-mode register used by DDR SDRAMs. This is the SDRAM's extended mode register, not the SDRAMC's mode/extended-mode register (SDMR) defined in Section 19.4.1, "SDRAM Mode/Extended Mode Register (SDMR)." Refer to the SDRAM manufacturer's device data sheet to confirm correct settings.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	0	1	OPTION											DLL

Figure 19-11. Typical DDR Extended Mode Register

Table 19-14. Typical DDR Extended-Mode Register Field Descriptions

Field	Description
BA1–BA0	Bank address. These must be set to 01 to select the extended mode register.
A11–A1 OPTION	Option. These bits are not defined by the DDR specification. Each DDR SDRAM manufacturer can use these bits to implement optional features. Check with the SDRAM manufacturer to determine if any optional features have been implemented. For normal operation all bits must be cleared.
A0 DLL	Delay locked loop. Controls enabling of the delay locked loop circuitry used for DDR timing. 0 Enabled 1 Disabled

19.5.1.6.3 Low-Power/Mobile DDR Extended Mode Register Definition

Figure 19-12 shows a typical extended-mode register used by low-power/mobile DDR SDRAMs. This is the SDRAM's extended mode register, not the SDRAMC's mode/extended-mode register (SDMR) defined in Section 19.4.1, "SDRAM Mode/Extended Mode Register (SDMR)." Refer to the SDRAM manufacturer's device data sheet to confirm correct settings.

	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Field	1	0	—					DS		TCSR		PASR		

Figure 19-12. Typical Mobile DDR Extended Mode Register

Table 19-15. Mobile DDR Extended-Mode Register Field Descriptions

Field	Description
BA1–BA0	Bank address. These must be set to 10 to select the extended mode register.
A11–A7	Reserved, must be cleared.
A6–A5 DS	Drive strength. 00 Full strength 01 Half strength 10 Quarter strength 11 One-eighth strength
A4–A3 TCSR	Temperature-compensated self-refresh. Check the SDRAM manufacturer's spec because the use of the TCSR settings can vary from memory to memory.
A2–A0 PASR	Partial array self refresh coverage. 000 Full array 001 Half array 010 Quarter array 101 One-eighth array 110 One-sixteenth array All other settings are reserved.

19.5.1.7 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.

19.5.1.8 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.

19.5.2 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 19-13 displays a simple timing diagram that illustrates the end result of the RCR delay.

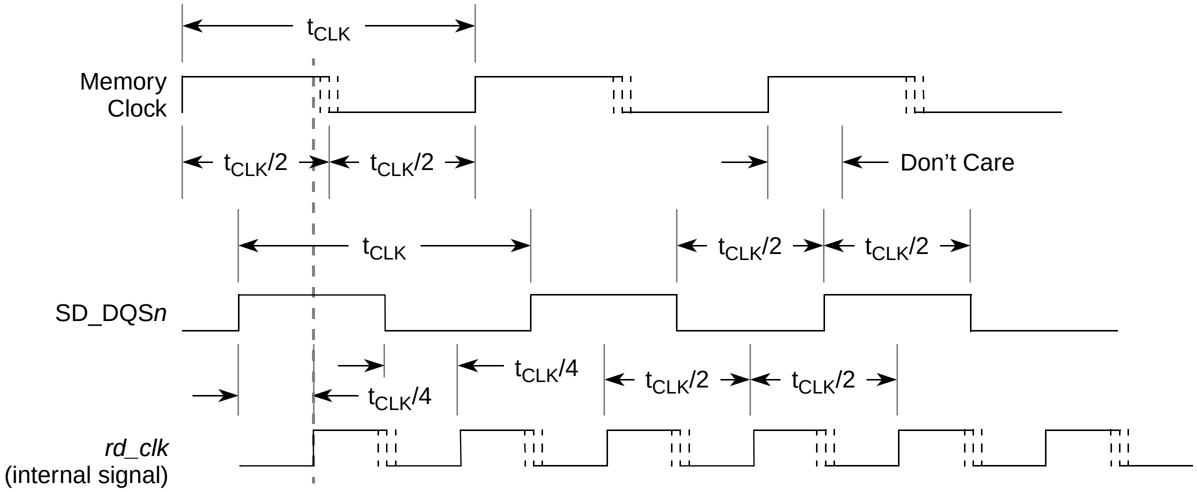


Figure 19-13. Frequency Doubler Block Diagram

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.

19.6 Initialization/Application Information

SDRAMs have a prescribed initialization sequence. The following sections detail the memory initialization steps for SDR, DDR SDRAM, and mobile DDR SDRAM. The sequence might change slightly from device-to-device. Refer to the SDRAM manufacturer's device datasheet as the most relevant reference.

19.6.1 SDR SDRAM Initialization Sequence

1. After reset is deactivated, pause for the amount of time indicated in SDRAM specification. Usually 100µs.
2. Configure pin multiplex control for shared $\overline{\text{SD\_CS}}$ pins in pin multiplexing and control module if needed.
3. pin multiplexing and controlWrite the SDCS*n* register values for each chip select that is used.
4. Program SDRAM configuration registers (SDCFG1 and SDCFG2) with correct delay and timing values.
5. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
6. Refresh the SDRAM. The SDRAM specification should indicate a number of refresh cycles to be performed before issuing an LMR command (usually two). Write to the SDCR with the IREF and MODE_EN bits set (SDCR[REF and IPALL] must be cleared). This forces a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.
7. Initialize the SDRAM's mode register using the LMR command. See Section 19.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LMR command.
8. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

19.6.2 DDR SDRAM Initialization Sequence

1. After reset is deactivated, pause for the amount of time indicated in SDRAM specification. Usually 200µs.
2. Configure pin multiplex control for shared $\overline{\text{SD\_CS}}$ pins in pin multiplexing and control module if needed.
3. pin multiplexing and controlWrite the SDCS*n* register values for each chip select that is used.

4. Program SDRAM configuration registers (SDCFG1 and SDCFG2) with correct delay and timing values.
5. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
6. Initialize the SDRAM's extended mode register to enable the DLL. See Section 19.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for instructions on issuing a LEMR command.
7. Initialize the SDRAM's mode register and reset the DLL using the LMR command. See Section 19.5.1.6, "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.
8. Pause for the DLL lock time specified by the memory.
9. Issue a second PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
10. Refresh the SDRAM. The SDRAM specification should indicate a number of refresh cycles to be performed before issuing an LMR command (usually two). Write to the SDCR with the IREF and MODE_EN bits set (SDCR[REF and IPALL] must be cleared). This forces a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.
11. Initialize the SDRAM's mode register using the LMR command. See Section 19.5.1.6, "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.
12. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

19.6.3 Low-power/Mobile SDRAM Initialization Sequence

1. After reset is deactivated, pause for the amount of time indicated in SDRAM specification. Usually 100µs or 200µs.
2. Configure pin multiplex control for shared $\overline{\text{SD\_CS}}$ pins in pin multiplexing and control module.
3. pin multiplexing and controlWrite the base address and mask registers (SDBAR0, SDBAR1, SDMR0, and SDMR1) to setup the address space for each chip-select.
4. Program SDRAM configuration registers (SDCFG1 and SDCFG2) with correct delay and timing values.
5. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] and SDCR[MODE_EN] set. The SDCR[REF and IREF] bits should remain cleared for this step.
6. Refresh the SDRAM. The SDRAM specification should indicate a number of refresh cycles to be performed before issuing an LMR command (usually two). Write to the SDCR with the IREF and MODE_EN bits set (SDCR[REF and IPALL] must be cleared). This forces a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.

7. Initialize the SDRAM's mode register using the LMR command. See Section 19.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LMR command.
8. Initialize the SDRAM's extended mode register using the LEMR command. See Section 19.5.1.6, "Load Mode/Extended Mode Register Command (lmr, lemr)," for more instruction on issuing an LEMR command.
9. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

19.6.4 Page Management

SDRAM devices have four internal banks. A particular row and bank of memory must be activated to allow read and write accesses. The SDRAM controller supports paging mode to maximize the memory access throughout. During operation, the SDRAM controller maintains an open page for each $\overline{\text{SD\_CS}}$ block. An open page is composed of the active rows in the internal banks. Each internal bank has its own active row.

The physical page size of a $\overline{\text{SD\_CS}}$ block is equal to the space size divided by the number of rows; but the page may not be contiguous in the internal address space because SDRAMs can have a different row address open in each bank and the internal address bits (A[27:24] and A[9:2]) or (A[27:24] and A[9:1]) used for memory column addresses are not consecutive.

Because the column address may split across two portions of the internal address, the contiguous page size is (number of contiguous columns per bank) × (number of bits). This gives a contiguous page size of 1 KBytes. However, the total (possibly fragmented) page size is (number of banks) × (number of columns) × (number of bits).

If a new access does not fall in the open row of an open bank of a $\overline{\text{SD\_CS}}$ block, the open row must be closed (PRE) and the new row must be opened (ACTV), then the READ or WRITE command can proceed. An ACTV command activates only one bank at one time. If another read or write falls in an inactive bank, another ACTV is needed, but no precharge is needed. If a read or write falls in any open row of any active banks of a page, no PRE or ACTV is needed; the read or write command can be issued immediately.

A page is kept open until one of the following conditions occurs:

- An access outside the open page.
- A refresh cycle is started.

All $\overline{\text{SD\_CS}}$ blocks are refreshed at the same time. The refresh closes all banks of every SDRAM block.

19.6.5 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 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 . 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.

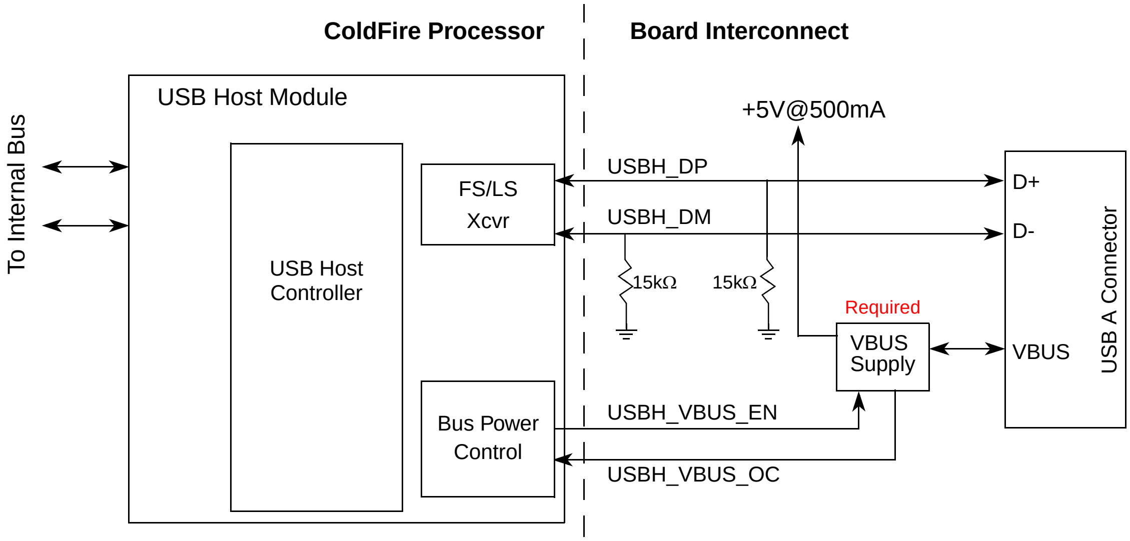


Figure 20-1. USB Host Interface Block Diagram

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
  - 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 9.3.7, “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
Drive VBUS	DRVVBUS	Enables drive of the 5 V power on VBUS	R/W	N
VBUS Power Fault	PWRFLT	Indicates a power fault occurred on VBUS (overcurrent)	R/W	N
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	0x0042_FA05	21.3.1.1/21-8
0xFC0B_4004	General Hardware Parameters (HWGENERAL)	N	32	R	0x0000_02C5	21.3.1.2/21-8
0xFC0B_4008	Host Hardware Parameters (HWHOST)	N	32	R	0x1002_0001	21.3.1.3/21-9
0xFC0B_4010	TX Buffer Hardware Parameters (HWTXBUF)	N	32	R	0x8004_0404	21.3.1.5/21-10
0xFC0B_4014	RX Buffer Hardware Parameters (HWRXBUF)	N	32	R	0x0000_0404	21.3.1.6/21-11
Capability Registers						
0xFC0B_4100	Host Interface Version Number (HCIVERSION)	Y	16	R	0x0100	21.3.3.1/21-13
0xFC0B_4103	Capability Register Length (CAPLENGTH)	Y	8	R	0x40	21.3.3.4/21-15
0xFC0B_4104	Host Structural Parameters (HCSPARAMS)	Y	32	R	0x0001_0011	21.3.3.3/21-14
0xFC0B_4108	Host Capability Parameters (HCCPARAMS)	Y	32	R	0x0000_0006	21.3.3.4/21-15
Operational Registers						

Table 20-3. USB Host Controller Memory Map (continued)

Address	Register	EHCI[1]	Width (bits)	Access	Reset	Section/Page
0xFC0B_4140	USB Command (USBCMD)	Y	32	R/W	0x0008_0B00	21.3.4.1/21-17
0xFC0B_4144	USB Status (USBSTS)	Y	32	R/W	0x0000_1000	21.3.4.2/21-19
0xFC0B_4148	USB Interrupt Enable (USBINTR)	Y	32	R/W	0x0000_0000	21.3.4.3/21-22
0xFC0B_414C	USB Frame Index (FRINDEX)	Y	32	R/W	0x0000_0000	21.3.4.4/21-24
0xFC0B_4154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	32	R/W	0x0000_0000	21.3.4.5/21-25
0xFC0B_4158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	32	R/W	0x0000_0000	21.3.4.7/21-26
0xFC0B_415C	Host TT Asynchronous Buffer Control (TTCTRL)	N	32	R/W	0x0000_0000	21.3.4.9/21-27
0xFC0B_4160	Master Interface Data Burst Size (BURSTSIZE)	N	32	R/W	0x0000_1010	21.3.4.10/21-28
0xFC0B_4164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	32	R/W	0x0002_0000	21.3.4.11/21-28
0xFC0B_4180	Configure Flag Register (CONFIGFLAG)	Y	32	R	0x0000_0001	21.3.4.12/21-30
0xFC0B_4184	Port Status/Control (PORTSC1)	Y	32	R/W	0xEC00_0000	21.3.4.13/21-30
0xFC0B_41A8	USB Mode Register (MODE)	N	32	R/W	0x0000_0003	21.3.4.15/21-37

[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*

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 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. See Chapter 9, “Chip Configuration Module (CCM),” for more information.

The USB controller provides control and status signals to interface with external USB OTG and USB host power devices. Use these control and status signals on the chip interface and the I^{2}C bus to communicate with external USB On-The-Go and USB host power devices.

USB-host modules must supply 500 mA with a 5 V supply on its downstream port (referred to as VBUS); however, the USB OTG standard provides a minimum 8 mA VBUS supply requirement. Optionally, the OTG module may supply up to 500 mA to the USB-connected devices. If the connected device attempts to draw more than the allocated amount of current, the USB host must disable the port and remove power. USB VBUS is not provided on-chip. This processor provides pins for control and status to an external IC capable of managing the VBUS downstream supply.

For OTG operations, external circuitry is required to manage the host negotiation protocol (HNP) and session request protocol (SRP). External ICs that are capable of providing the OTG VBUS with support for HNP and SRP, as well as support for programmable pullup and pulldown resistors on the USB DP and DM lines are available from various manufacturers.

The on-chip FS/LS transceiver also includes a programmable pullup resistor on USB DP. This pullup is configurable via the CCM. Also, configurable 50 kΩ pulldown resistors are available on the USB_DP and USB_DM signals of the on-chip transceiver. See Chapter 9, “Chip Configuration Module (CCM),” for more information. The primary function of the transceiver is the physical signal conditioning of the external USB DP and DM cable signals for a USB 2.0 network. Several USB system elements are not supported on the device as they are available via standard products from various manufacturers.

21.1.2 Block Diagram

Figure 21-1 shows the USB On-The-Go interface using the on-chip full-speed/low-speed transceiver.

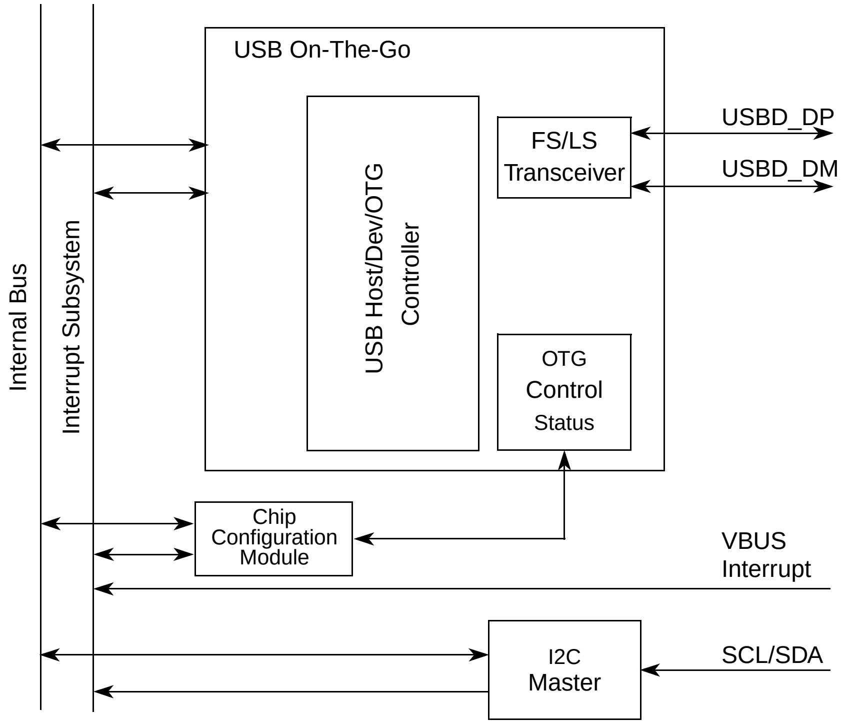


Figure 21-1. USB On-The-Go with on-chip FS/LS Transceiver Interface Block Diagram

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 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

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.

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.

Table 21-2. USB OTG Signal Descriptions

Signal	I/O	Description	
On-chip FS/LS transceiver			
USB_CLKIN	I	Optional 60 MHz clock source.	
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
USB_PULLUP	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
USB_VBUS_EN	O	Enables the off-chip VBUS charge pump when USB OTG module is configured as a host.	
		State Meaning	Asserted—Enables the external VBUS supply. Negated—Disables the external VBUS supply.
		Timing	Asynchronous
USB_VBUS_OC	I	Alerts the processor that a short (overcurrent) has occurred on USB data bus.	
		State Meaning	Asserted—Power fault occurred. Negated—No power fault.
		Timing	Asynchronous

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^{2}C bus or GPIO pins.

The OTG controller status register (UOCSR) implements as follows:

- Writes to the UOCSR register from the firmware set the corresponding bits on the USB interface.
- When the USB OTG module outputs change, the corresponding bits on the UOCSR register are updated, and a maskable interrupt is generated.

The UOCSR register is documented in the CCM chapter, see Section 9.3.6, "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?
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

Table 21-3. Internal Control and Status Bits for USB OTG Module (continued)

Signal	Mnemonic	Direction	Comments	Interrupt Trigger?
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	0x0042_FA05	21.3.1.1/21-8
0xFC0B_0004	General Hardware Parameters (HWGENERAL)	N	H/D	32	R	0x0000_07C5	21.3.1.2/21-8
0xFC0B_0008	Host Hardware Parameters (HWHOST)	N	H/D	32	R	0x1002_0001	21.3.1.3/21-9
0xFC0B_000C	Device Hardware Parameters (HWDEVICE)	N	D	32	R	0x0000_0009	21.3.1.4/21-9
0xFC0B_0010	TX Buffer Hardware Parameters (HWTXBUF)	N	H/D	32	R	0x8004_0604	21.3.1.5/21-10
0xFC0B_0014	RX Buffer Hardware Parameters (HWRXBUF)	N	H/D	32	R	0x0000_0404	21.3.1.6/21-11
Device/Host Timer Registers							
0xFC0B_0080	General Purpose Timer 0 Load (GPTIMER0LD)	N	H/D	32	R/W	0x0000_0000	21.3.2.1/21-11
0xFC0B_0084	General Purpose Timer 0 Control (GPTIMER0CTL)	N	H/D	32	R/W	0x0000_0000	21.3.2.2/21-12
0xFC0B_0088	General Purpose Timer 1 Load (GPTIMER1LD)	N	H/D	32	R/W	0x0000_0000	21.3.2.1/21-11
0xFC0B_008C	General Purpose Timer 1 Control (GPTIMER1CTL)	N	H/D	32	R/W	0x0000_0000	21.3.2.2/21-12
Capability Registers							
0xFC0B_0100	Host Interface Version Number (HCIVERSION)	Y	H	16	R	0x0100	21.3.3.1/21-13
0xFC0B_0103	Capability Register Length (CAPLENGTH)	Y	H/D	8	R	0x40	21.3.3.2/21-13

Table 21-4. USB On-The-Go Memory Map (continued)

Address	Register	EHCI[1]	H/D[2]	Width (bits)	Access	Reset	Section/Page
0xFC0B_0104	Host Structural Parameters (HCSPARAMS)	Y	H	32	R	0x0001_0011	21.3.3.3/21-14
0xFC0B_0108	Host Capability Parameters (HCCPARAMS)	Y	H	32	R	0x0000_0006	21.3.3.4/21-15
0xFC0B_0122	Device Interface Version Number (DCIVERSION)	N	D	16	R	0x0001	21.3.3.5/21-15
0xFC0B_0124	Device Capability Parameters (DCCPARAMS)	N	D	32	R	0x0000_0184	21.3.3.6/21-16
Operational Registers							
0xFC0B_0140	USB Command (USBCMD)	Y	H/D	32	R/W	0x0008_0000	21.3.4.1/21-17
0xFC0B_0144	USB Status (USBSTS)	Y	H/D	32	R/W	0x0000_0080	21.3.4.2/21-19
0xFC0B_0148	USB Interrupt Enable (USBINTR)	Y	H/D	32	R/W	0x0000_0000	21.3.4.3/21-22
0xFC0B_014C	USB Frame Index (FRINDEX)	Y	H/D	32	R/W	0x0000_0000	21.3.4.4/21-24
0xFC0B_0154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	H	32	R/W	0x0000_0000	21.3.4.5/21-25
0xFC0B_0154	Device Address (DEVICEADDR)	N	D	32	R/W	0x0000_0000	21.3.4.6/21-26
0xFC0B_0158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	H	32	R/W	0x0000_0000	21.3.4.7/21-26
0xFC0B_0158	Address at Endpoint List (EPLISTADDR)	N	D	32	R/W	0x0000_0000	21.3.4.8/21-27
0xFC0B_015C	Host TT Asynchronous Buffer Control (TTCTRL)	N	H	32	R/W	0x0000_0000	21.3.4.9/21-27
0xFC0B_0160	Master Interface Data Burst Size (BURSTSIZE)	N	H/D	32	R/W	0x0000_0404	21.3.4.10/21-28
0xFC0B_0164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	H	32	R/W	0x0000_0000	21.3.4.11/21-28
0xFC0B_0180	Configure Flag Register (CONFIGFLAG)	Y	H/D	32	R	0x0000_0001	21.3.4.12/21-30
0xFC0B_0184	Port Status/Control (PORTSC1)	Y	H/D	32	R/W	0xEC00_0004	21.3.4.13/21-30
0xFC0B_01A4	On-The-Go Status and Control (OTGSC)	N	H/D	32	R/W	0x0000_1020	21.3.4.14/21-35
0xFC0B_01A8	USB Mode Register (MODE)	N	H/D	32	R/W	0x0000_0000	21.3.4.15/21-37
0xFC0B_01AC	Endpoint Setup Status Register (EPSETUPSR)	N	D	32	R/W	0x0000_0000	21.3.4.16/21-39
0xFC0B_01B0	Endpoint Initialization (EPPRIME)	N	D	32	R/W	0x0000_0000	21.3.4.17/21-39
0xFC0B_01B4	Endpoint De-initialize (EPFLUSH)	N	D	32	R/W	0x0000_0000	21.3.4.18/21-40
0xFC0B_01B8	Endpoint Status Register (EPSR)	N	D	32	R	0x0000_0000	21.3.4.19/21-40
0xFC0B_01BC	Endpoint Complete (EPCOMPLETE)	N	D	32	R/W	0x0000_0000	21.3.4.20/21-41
0xFC0B_01C0	Endpoint Control Register 0 (EPCR0)	N	D	32	R/W	0x0080_0080	21.3.4.21/21-42
0xFC0B_01C4	Endpoint Control Register 1 (EPCR1)	N	D	32	R/W	0x0000_0000	21.3.4.22/21-43
0xFC0B_01C8	Endpoint Control Register 2 (EPCR2)	N	D	32	R/W	0x0000_0000	21.3.4.22/21-43
0xFC0B_01CC	Endpoint Control Register 3 (EPCR3)	N	D	32	R/W	0x0000_0000	21.3.4.22/21-43

[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	1	0	1	1	1	1	1	0	1	0	0	0	0	0	0	1	0	1

Figure 21-2. 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 21-3. 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 21-4. 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	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 21-5. 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 21-6. 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 21-7. 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 Device/Host Timer Registers

The host/device controller drivers can measure time-related activites using these timer registers, which are not defined by the EHCI specification.

21.3.2.1 General Purpose Timer *n* Load Registers (GPTIMER*n*LD)

The GPTIMER*n*LD registers contain the timer duration or load value.

Address: 0xFC0B_0080 (GPTIMER0LD)
0xFC0B_0088 (GPTIMER1LD)

Access: User read/write

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

Figure 21-8. General Purpose Timer *n* Load Registers (GPTIMER*n*LD)

Table 21-11. GPTIMER*n*LD Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–0 GPTLD	Specifies the value to be loaded into the countdown timer on a reset. The value in this register represents the time in microseconds minus 1 for the timer duration. For example, for a one millisecond timer, load 1000 – 1 = 999 (0x00_03E7). **Note:** Maximum value of 0xFF_FFFF or 16.777215 seconds.

21.3.2.2 General Purpose Timer *n* Control Registers (GPTIMER*n*CTL)

The GPTIMER*n*CTL registers control the various functions of the general purpose timers.

Address: 0xFC0B_0084 (GPTIMER0CTL)
0xFC0B_008C (GPTIMER1CTL)

Access: User read/write

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

Figure 21-9. General Purpose Timer *n* Control Registers (GPTIMER*n*CTL)

Table 21-12. GPTIMER*n*CTL Field Descriptions

Field	Description
31 RUN	Timer run. Enables the general purpose timer. Setting or clearing this bit does not have an effect on the GPTCNT field. 0 Timer stop 1 Timer run
30 RST	Timer reset. Setting this bit reloads GPTCNT with the value in GPTIMER*n*LD[GPTLD]. 0 No action 1 Load counter value
29–25	Reserved, must be cleared.
24 MODE	Timer mode. Selects between a single timer countdown and a looped countdown. In one-shot mode, the timer counts down to zero, generates an interrupt, and stops until the counter is reset by software. In repeat mode, the timer counts down to zero, generates an interrupt, and automatically reloads the counter and begins another countdown. 0 One shot 1 Repeat
23–0 GPTCNT	Timer count. Indicates the current value of the running timer.

21.3.3 Capability Registers

Specifies software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers not defined by the EHCI specification are noted in their descriptions.

21.3.3.1 Host Controller Interface Version Register (HCIVERSION)

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this OTG controller. The most-significant byte of the register represents a major revision; the least-significant byte is the minor revision. Figure 21-10 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 21-10. Host Controller Interface Version Register (HCIVERSION)

Table 21-13. HCIVERSION Field Descriptions

Field	Description
15–0 HCIVERSION	EHCI revision number. Value is 0x0100 indicating version 1.0.

21.3.3.2 Capability Registers Length Register (CAPLENGTH)

Register is used as an offset to add to the register base address to find the beginning of the operational register space, the location of the USBCMD register.

Address: 0xFC0B_0103 (CAPLENGTH) Access: User read-only

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

Figure 21-11. Capability Registers Length Register (CAPLENGTH)

Table 21-14. CAPLENGTH Field Descriptions

Field	Description
7–0 CAPLENGTH	Capability registers length. Always 0x40.

21.3.3.3 Host Controller Structural Parameters Register (HCSPARAMS)

This register contains structural parameters such as the number of downstream ports. Figure 21-12 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	See Note	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

Note: Set on USB host module; Cleared on USB OTG.

Figure 21-12. Host Controller Structural Parameters Register (HCSPARAMS)

Table 21-15. HCSPARAMS Field Descriptions

Field	Description
31–28	Reserved, always cleared.
27–24 N_TT	Number of transaction translators. Non-EHCI field. Indicates number of embedded transaction translators associated with host controller. This field is always 0x0. See Section 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 on USB host, cleared on USB OTG. 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.

21.3.3.4 Host Controller Capability Parameters Register (HCCPARAMS)

Identifies multiple mode control (time-base bit functionality) addressing capability.

Address: 0xFC0B_0108 (HCCPARAMS) Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EECP								IST				0	ASP	PFL	ADC
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 21-13. Host Controller Capability Parameters Register (HCCPARAMS)

Table 21-16. HCCPARAMS Field Descriptions

Field	Description
31–16	Reserved, always cleared.
15–8 EECP	EHCI extended capabilities pointer. This optional field indicates the existence of a capabilities list. 0x00 No extended capabilities are implemented. This field is always 0.
7–4 IST	Isochronous scheduling threshold. Indicates where software can reliably update the isochronous schedule, relative to the current position of the executing host controller. This field is always 0. 0 The value of the least significant 3 bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state.
3	Reserved, always cleared.
2 ASP	Asynchronous schedule park capability. Indicates if the host controller supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. This bit is always set. 0 Park not supported. 1 Park supported.
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

21.3.3.5 Device Controller Interface Version (DCIVERSION)

Not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision.

Address: 0xFC0B_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 21-14. Device Controller Interface Version Register (DCIVERSION)

Table 21-17. DCIVERSION Field Descriptions

Field	Description
15–0 DCIVERSION	Device interface revision number.

21.3.3.6 Device Controller Capability Parameters (DCCPARAMS)

Not defined in the EHCI specification. Register describes the overall host/device capability of the USB OTG module.

Address: 0xFC0B_0124 (DCCPARAMS) Access: User read-only

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

Figure 21-15. Device Control Capability Parameters (DCCPARAMS)

Table 21-18. DCCPARAMS Field Descriptions

Field	Description
31–9	Reserved, always cleared.
8 HC	Host capable. Indicates the USB OTG controller can operate as an EHCI compatible USB 2.0 host. Always set.
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.

21.3.4 Operational Registers

Comprised of dynamic control or status registers and are defined below.

21.3.4.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Address: 0xFC0B_0140 (USBCMD) Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FS2	ATDTW	SUTW	0	ASPE	0	ASP		0	IAA	ASE	PSE	FS1	FS0	RST	RS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-16. USB Command Register (USBCMD)

Table 21-19. USBCMD Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–16 ITC	Interrupt threshold control. System software uses this field to set the maximum rate at which the module issueS interrupts. ITC contains maximum interrupt interval measured in microframes. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 64 microframes Else Reserved
15 FS2	See the FS bit description below. This is a non-EHCI bit.
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 21-19. 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 21-19. USBCMD Field Descriptions (continued)

Field	Description
1 RST	Controller reset. Software uses this bit to reset controller. Controller clears this bit when reset process completes. Clearing this register does not allow software to terminate the reset process early. Host mode (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.

21.3.4.2 USB Status Register (USBSTS)

This register indicates various states of each module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them.

Address: 0xFC0B_0144 (USBSTS)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	TI1	TI0	0	0	0	0	UPI	UAI	0	NAKI
W							w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	AS	PS	RCL	HCH	0	0	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 21-17. USB Status Register (USBSTS)

Table 21-20. USBSTS Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25 TI1	General purpose timer 1 interrupt. Set when the counter in the GPTIMER1CTRL register transitions to zero. Writing a one to this bit clears it. 0 No interrupt 1 Interrupt occurred.
24 TI0	General purpose timer 0 interrupt. Set when the counter in the GPTIMER0CTRL register transitions to zero. Writing a one to this bit clears it. 0 No interrupt 1 Interrupt occurred.
23–20	Reserved, must be cleared.
19 UPI	USB host periodic interrupt. Set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the periodic schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the periodic schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. **Note:** This bit is not used by the device controller and is always zero.
18 UAI	USB host asynchronous interrupt. Set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the asynchronous schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the asynchronous schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. **Note:** This bit is not used by the device controller and is always zero.
17	Reserved, must be cleared.
16 NAKI	NAK interrupt. Set by hardware for a particular endpoint when the TX/RX endpoint’s NAK bit and the corresponding TX/RX endpoint’s NAK enable bit are set. The hardware automatically clears this bit when all the enabled TX/RX endpoint NAK bits are cleared.
15 AS	Asynchronous schedule status. Reports the current real status of asynchronous schedule. Controller is not immediately required to disable or enable the asynchronous schedule when software transitions the USBCMD[ASE] bit. When this bit and the USBCMD[ASE] bit have the same value, the asynchronous schedule is enabled (1) or disabled (0). Used only in host mode. 0 Disabled. 1 Enabled.
14 PS	Periodic schedule status. Reports current real status of periodic schedule. Controller is not immediately required to disable or enable the periodic schedule when software transitions the USBCMD[PSE] bit. When this bit and the USBCMD[PSE] bit have the same value, the periodic schedule is enabled or disabled. Used only in host mode. 0 Disabled. 1 Enabled.
13 RCL	Reclamation. DetectS an empty asynchronous schedule. Used only by the host mode. 0 Non-empty asynchronous schedule. 1 Empty asynchronous schedule.

Table 21-20. USBSTS Field Descriptions (continued)

Field	Description
12 HCH	Host controller halted. This bit is cleared when the USBCMD[RS] bit is set. The controller sets this bit after it stops executing because of the USBCMD[RS] bit being cleared, by software or the host controller hardware (for example, internal error). Used only in host mode. 0 Running. 1 Halted.
11–9	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.
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.

Table 21-20. USBSTS Field Descriptions (continued)

Field	Description
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 21-57 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.

21.3.4.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt generates when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) continues to show interrupt sources (even if the USBINTR register disables them), allowing polling of interrupt events by the software.

Address: 0xFC0B_0148 (USBINTR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	TIE1	TIE0	0	0	0	0	UPIE	UAIE	0	NAKE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	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 21-18. USB Interrupt Enable Register (USBINTR)

Table 21-21. USBINTR Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25 TIE1	General purpose timer 1 interrupt enable. When this bit and USBSTS[GPTINT1] are set, the USB controller issues an interrupt to the processor. The interrupt is acknowledged by clearing GPTINT1. 0 Disabled 1 Enabled
24 TIE0	General purpose timer 0 interrupt enable. When this bit and USBSTS[GPTINT0] are set, the USB controller issues an interrupt to the processor. The interrupt is acknowledged by clearing GPTINT0. 0 Disabled 1 Enabled
23–20	Reserved, must be cleared.
19 UPIE	USB host periodic interrupt enable. When this bit and USBSTS[USBHSTPERINT] are set, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing USBHSTPERINT.
18 UAIE	USB host asynchronous interrupt enable. When this bit and USBSTS[USBHSTASYNCINT] are set, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing USBHSTASYNCINT.
17	Reserved, must be cleared.
16 NAKE	NAK interrupt enable. When this bit and the USBSTS[NAKI] bit are set, an interrupt generates. 0 Disabled 1 Enabled
15–9	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 21-21. 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

21.3.4.4 Frame Index Register (FRINDEX)

In host mode, the controller uses this register to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits [N–3] select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the USBCMD[FS] field.

This register must be a longword. Byte writes produce undefined results. This register cannot be written unless the USB OTG controller is in halted state as the USBSTS[HCH] bit indicates. A write to this register while the USBSTS[RS] bit is set produces undefined results. Writes to this register also affect the SOF value.

In device mode (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 21-19. Frame Index Register (FRINDEX)

Table 21-22. FRINDEX Field Descriptions

Field	Description
31–14	Reserved, must be cleared.
13–0 FRINDEX	Frame index. The value in this register increments at the end of each time frame (microframe). Bits [N– 3] are for the frame list current index. This means each location of the frame list is accessed 8 times per frame (once each microframe) before moving to the next index. In device mode 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 21-23 illustrates values of N based on the value of the USBCMD[FS] field when used in host mode.

Table 21-23. FRINDEX N Values

USBCMD[FS]	Frame List Size	FRINDEX N value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

21.3.4.5 Periodic Frame List Base Address Register (PERIODICLISTBASE)

This register contains the beginning address of the periodic frame list in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer assumes to be 4-Kbyte aligned. The contents combine with the FRINDEX register to enable the controller to step through the periodic frame list in sequence.

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 21.3.4.6, “Device Address Register (DEVICEADDR),” for more information.

Address: 0xFC0B_0154 (PERIODICLISTBASE)

Access: User read/write

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

Figure 21-20. Periodic Frame List Base Address Register (PERIODICLISTBASE)

Table 21-24. PERIODICLISTBASE Field Descriptions

Field	Description
31–12 PERBASE	Base Address. These bits correspond to memory address signal [31:12]. Used only in the host mode
11–0	Reserved, must be cleared.

21.3.4.6 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 21.3.4.5, “Periodic Frame List Base Address Register (PERIODICLISTBASE),” for more information.

Address: 0xFC0B_0154 (DEVICEADDR) Access: User read/write

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

Figure 21-21. Device Address Register (DEVICEADDR)

Table 21-25. DEVICEADDR Field Descriptions

Field	Description
31–25 USBADR	Device Address. This field corresponds to the USB device address.
24–0	Reserved, must be cleared.

21.3.4.7 Current Asynchronous List Address Register (ASYNCLISTADDR)

The ASYNCLISTADDR register contains the address of the next asynchronous queue head to executed by the host.

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 21.3.4.8, “Endpoint List Address Register (EPLISTADDR),” for more information.

Address: 0xFC0B_0158 (ASYNCLISTADDR) Access: User read/write

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

Figure 21-22. Current Asynchronous List Address Register (ASYNCLISTADDR)

Table 21-26. ASYNCLISTADDR Field Descriptions

Field	Description
31–5 ASYBASE	Link pointer low (LPL). These bits correspond to memory address signal [31:5]. This field may only reference a queue head (QH). Used only in host mode.
4–0	Reserved, must be cleared.

21.3.4.8 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 Section 21.3.4.7, "Current Asynchronous List Address Register (ASYNCLISTADDR)," for more information.

Address: 0xFC0B_0158 (EPLISTADDR) Access: User read/write

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

Figure 21-23. Endpoint List Address Register (EPLISTADDR)

Table 21-27. EPLISTADDR Field Descriptions

Field	Description
31–11 EPBASE	Endpoint list address. Correspond to memory address signals [31:11] References a list of up to 32 queue heads (i.e. one queue head per endpoint and direction). Address of the top of the endpoint list.
10–0	Reserved, must be cleared.

21.3.4.9 Host TT Asynchronous Buffer Control (TTCTRL)

Address: 0xFC0B_015C (TTCTRL) Access: User read/write

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

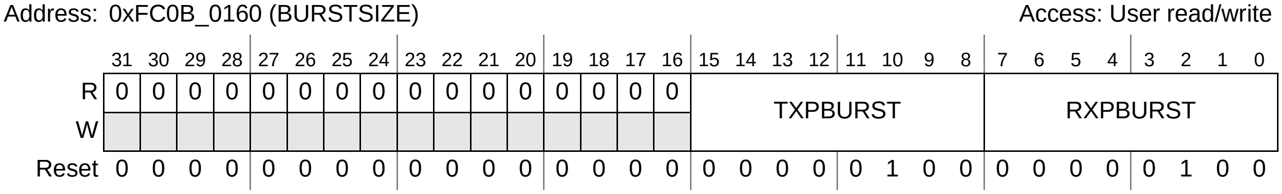
Figure 21-24. Host TT Asynchronous Buffer Control (TTCTRL)

Table 21-28. TTCTRL Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–24 TTHA	TT Hub Address. This field is used to match against the Hub Address field in a QH or siTD to determine if the packet is routed to the internal TT for directly attached FS/LS devices. If the hub address in the QH or siTD does not match this address then the packet is broadcast on the high speed ports destined for a downstream HS hub with the address in the QH or siTD.
23–0	Reserved, must be cleared.

21.3.4.10 Master Interface Data Burst Size Register (BURSTSIZE)

This register is not defined in the EHCI specification. BURSTSIZE dynamically controls the burst size during data movement on the initiator (master) interface.



Address: 0xFC0B_0160 (BURSTSIZE) Access: User read/write

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

Figure 21-25. Master Interface Data Burst Size (BURSTSIZE)

Table 21-29. BURSTSIZE Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 TXPBURST	Programable TX burst length. Represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater than 16.
7–0 RXPBURST	Programable RX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16.

21.3.4.11 Transmit FIFO Tuning Control Register (TXFILLTUNING)

This register is not defined in the EHCI specification. The TXFILLTUNING register controls performance tuning associated with how the module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include how much data to post into the FIFO and an estimate for how long that operation takes in the target system.

Definitions:

T_0 = Standard packet overhead

T_1 = Time to send data payload

T_s = Total packet flight time (send-only) packet ($T_s = T_0 + T_1$)

T_{ff} = Time to fetch packet into TX FIFO up to specified level

T_p = Total packet time (fetch and send) packet ($T_p = T_{ff} + T_s$)

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, the host controller checks to ensure T_p remains before the end of the (micro)frame. If so, it pre-fills the TX FIFO. If at anytime during the pre-fill operation the time remaining the (micro)frame is less than T_s, packet attempt ceases and tries at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to mark the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic beginning after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and should be minimized (not necessarily eliminated). The TSCHHEALTH (T_{ff}) parameter described below can minimize back-offs.

Address: 0xFC0B_0164 (TXFILLTUNING) Access: User read/write

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

Figure 21-26. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 21-30. TXFILLTUNING Field Descriptions

Field	Description
31–22	Reserved, must be cleared.
21–16 TXFIFOTHRES	FIFO burst threshold. Controls the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on the bus. The minimum value is 2 and this value should be as low as possible to maximize USB performance. Systems with unpredictable latency and/or insufficient bandwidth can use a higher value where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can replenish from system memory. This value is ignored if the USBMODE[SDIS] bit is set. When the USBMODE[SDIS] bit is set, the host controller behaves as if TXFIFOTHRES is set to its maximum value.
15–13	Reserved, must be cleared.

Table 21-30. TXFILLTUNING Field Descriptions (continued)

Field	Description
12–8 TXSCHHEALTH	Scheduler health counter. These bits increment when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next SOF. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter stops counting after reaching the maximum of 31.
7–0 TXSCHOH	Scheduler overhead. These bits add an additional fixed offset to the schedule time estimator described as T_{ff}. As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH field to less than 10 per second in a highly utilized bus. Choosing a value too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267 µs when a device connects in high-speed mode. The time unit represented in this register is 6.333 µs when a device connects in low-/full-speed mode. For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: $$\frac{\mathrm{TXFIFOTHRES} \times (\mathrm{BURSTSIZE} \times 4)}{40 \times TimeUnit}$$ ***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.

21.3.4.12 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 21-31. CONFIGFLAG Field Descriptions

Field	Description
31–0	Reserved. (0x0000_0001, all port routings default to this host)

21.3.4.13 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*)*

Table 21-32. 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 Reserved 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.

Table 21-32. PORTSC1 Field Descriptions (continued)

Field	Description
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.
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’s USB host module, the port indicator signals are implemented as status bits within the CCM. On the USB OTG module this feature is not implemented, therefore this field is read-only and is always cleared.
13 PO	Port owner. Port owner handoff is not implemented in this design, therefore this bit is read-only and is always cleared.
12 PP	Port power. Represents the current setting of the port power control switch (0 equals off, 1 equals on). When power is not available on a port (PP = 0), it is non-functional and does not report attaches, detaches, etc. When an over-current condition is detected on a powered port, the host controller driver from a 1to a 0 (removing power from the port) transitions the PP bit in each affected port.

Table 21-32. PORTSC1 Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>11–10
LS</td><td>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</td></tr>
<tr><td>9
HSP</td><td>High speed port. Indicates if the host/device connected is in high speed mode.
0 FS or LS
1 HS
<b>Note:</b> This bit is redundant with the PSPD bit field.</td></tr>
<tr><td>8
PR</td><td>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 <i>USB Specification Revision 2.0</i> 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.</td></tr>
<tr><td>7
SUSP</td><td>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:
<table>
<tr><th>PE</th><th>SUSP</th><th>Port State</th></tr>
<tr><td>0</td><td><i>x</i></td><td>Disable</td></tr>
<tr><td>1</td><td>0</td><td>Enable</td></tr>
<tr><td>1</td><td>1</td><td>Suspend</td></tr>
</table>
When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was set. In the suspend state, the port is sensitive to resume detection. The bit status does not change until the port is suspended and there may be a delay in suspending a port if there is a transaction currently in progress on the USB.
The module unconditionally clears this bit when software clears the FPR bit. The host controller ignores clearing this bit. If host software sets this bit when the port is not enabled (PE = 0), the results are undefined.
This field is cleared if the PP bit is cleared in host mode.
Device mode (USB OTG only):
In device mode, this bit is a read-only status bit.</td></tr>
</table>

Table 21-32. PORTSC1 Field Descriptions (continued)

Field	Description
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.
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).

Table 21-32. PORTSC1 Field Descriptions (continued)

Field	Description
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)

21.3.4.14 On-the-Go Status and Control Register (OTGSC)

This register is not defined in the EHCI specification. The host controller implements one OTGSC register corresponding to port 0 of the host controller.

The OTGSC register has four sections:

- OTG interrupt enables (read/write)
- OTG interrupt status (read/write to clear)
- OTG status inputs (read-only)
- OTG controls (read/write)

The status inputs de-bounce using a 1 ms time constant. Values on the status inputs that do not persist for more than 1 ms do not cause an update of the status inputs or an OTG interrupt.

Address: 0xFC0B_01A4 (OTGSC) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	DPIE	1MSE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	0	DPIS	1MSS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS
W										w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	DPS	1MST	BSE	BSV	ASV	AVV	ID	0	0	IDPU	DP	OT	0	VC	VD
W																
Reset	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Figure 21-29. On-the-Go Status and Control Register (OTGSC)

Table 21-33. 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.
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.

Table 21-33. OTGSC Field Descriptions (continued)

Field	Description
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.
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.

21.3.4.15 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 21-34. 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 21-34. 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.

21.3.4.16 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

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

Figure 21-31. Endpoint Setup Status Register (EPSETUPSR)

Table 21-35. 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.

21.3.4.17 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

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

Figure 21-32. Endpoint Initialization Register (EPPRIME)

Table 21-36. 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.

21.3.4.18 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

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

Figure 21-33. Endpoint Flush Register (EPFLUSH)

Table 21-37. 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.

21.3.4.19 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 21-38. 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.

21.3.4.20 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 21-39. 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 21-39. 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.

21.3.4.21 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 21-40. 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 21-40. 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

21.3.4.22 Endpoint Control Register *n* (EPCR*n*)

These registers are not defined in the EHCI specification. There is an EPCR*n* register for each endpoint in a device.

Address: 0xFC0B_01C4 (EPCR1)
0xFC0B_01C8 (EPCR2)
0xFC0B_01CA (EPCR3)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	TXE	0	TXI	0	TXT		TXD	TXS
W										TXR						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	RXE	0	RXI	0	RXT		RXD	RXS
W										RXR						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-37. Endpoint Control Registers (EPCR*n*)

Table 21-41. 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 21-41. 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.

Due to pin-count limitations the USB modules only support certain combinations of PHY interfaces and USB functionality. Refer to the Table 21-42 for more information.

Table 21-42. USB Network Speed and Required Physical Interface

USB Mode and Speed	DP and DN On-Chip Analog XCVR Active	I^2C	FEC	External Integrated Circuit Required
USB Host FS/LS	Yes	No	Yes	See Section 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”

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.

Also included is an internal 1.5k Ω pull-up resistor on DP controlled by the CCM. (See ,” for more details.) This allows the OTG module to operate in full-speed device operation. Host operation requires this internal resistor to be disabled via the CCM, and 15k Ω external resistors to connect from DP and DM signals to ground.

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.

To initialize the host controller, software must:

1. Optionally set streaming disable in the USBMODE[SDIS] bit.
2. Optionally modify the BURSTSIZE register.
3. Optionally write the appropriate value to the USBINTR register to enable the desired interrupts.
4. Set the USBMODE[CM] field to enable host mode, and set the USBMODE[ES] bit for big endian operation.
5. 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.
6. Enable external VBUS supply. The exact steps required for initialization depend on the external hardware used to supply the 5V VBUS power.
7. 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.

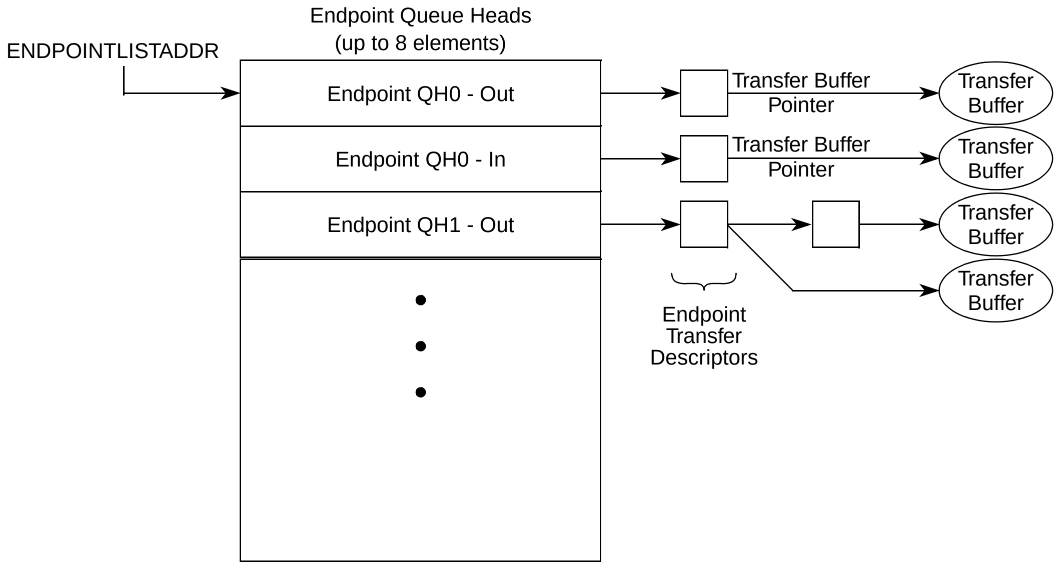


Figure 21-38. End Point Queue Head Organization

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 21-43. 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 21-43. 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 21-44. 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

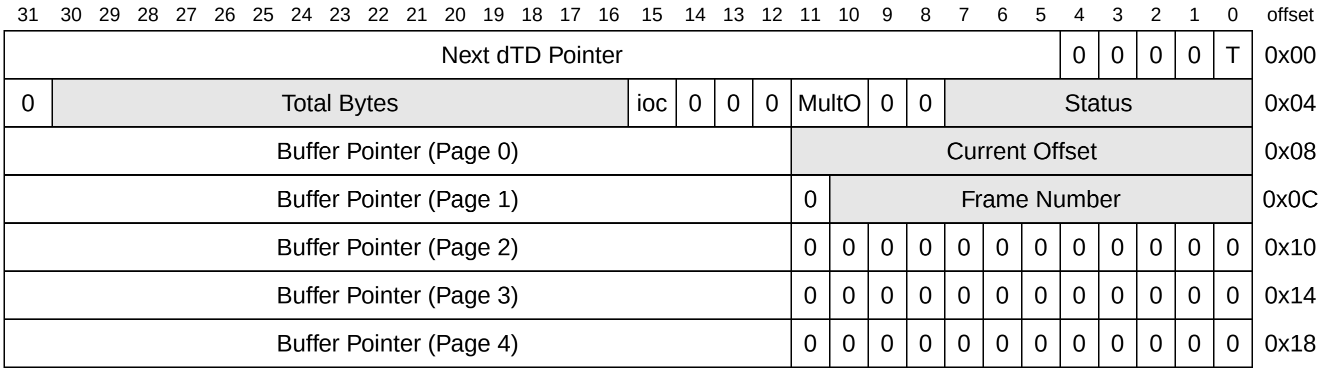
Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head receives setup data packets.

Table 21-45. 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.”



<table>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td>offset</td></tr>
<tr><td colspan="27">Next dTD Pointer</td><td>0</td><td>0</td><td>0</td><td>0</td><td>T</td><td>0x00</td></tr>
<tr><td>0</td><td colspan="15">Total Bytes</td><td>ioc</td><td>0</td><td>0</td><td>0</td><td colspan="2">MultO</td><td>0</td><td>0</td><td colspan="8">Status</td><td>0x04</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 0)</td><td colspan="12">Current Offset</td><td>0x08</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 1)</td><td>0</td><td colspan="11">Frame Number</td><td>0x0C</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 2)</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0x10</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 3)</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0x14</td></tr>
<tr><td colspan="20">Buffer Pointer (Page 4)</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0x18</td></tr>
</table>

▨ Device controller read/write; all others read-only.

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 21-46. 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 21-47. 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 21-47. dTD Token (continued)

<table>
<tr><th>Field</th><th colspan="2">Description</th></tr>
<tr><td>11–10
MultO</td><td colspan="2">Multiplier Override. This field can possibly transmit-ISOs (ISO-IN) to override the multiplier in the QH. This field must be 0 for all packet types not transmit-ISO.

For example, if QH.MULT equals 3; Maximum packet size equals 8; Total Bytes equals 15; MultiO equals 0 [default], then three packets are sent: {Data2(8); Data1(7); Data0(0)}.

If QH.MULT equals 3; Maximum packet size equals 8; Total Bytes equals 15; MultO equals 2, then two packets are sent: {Data1(8); Data0(7)}

For maximal efficiency, software must compute MultO equals greatest integer of (Total Bytes / Max. Packet Size) except for the case when Total Bytes equals 0; then MultO must be 1.

Note: Non-ISO and Non-TX endpoints must set MultO equals 00.</td></tr>
<tr><td>9–8</td><td colspan="2">Reserved. Reserved for future use and must be cleared.</td></tr>
<tr><td rowspan="8">7–0
Status</td><td colspan="2">Status. Device controller communicates individual command execution states back to the DCD software. This field contains the status of the last transaction performed on this dTD. The bit encodings are:</td></tr>
<tr><th>Bit</th><th>Status Field Description</th></tr>
<tr><td>7</td><td>Active. Set by software to enable the execution of transactions by the device controller.</td></tr>
<tr><td>6</td><td>Halted. Set by the device controller during status updates to indicate a serious error has occurred at the device/endpoint addressed by this dTD. Any time a transaction results in the halted bit being set, the active bit is also cleared.</td></tr>
<tr><td>5</td><td>Data Buffer Error. Set by the device controller during status update to indicate the device controller is unable to maintain the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run).</td></tr>
<tr><td>4</td><td>Reserved.</td></tr>
<tr><td>3</td><td>Transaction Error. Set by the device controller during status update in case the device did not receive a valid response from the host (time-out, CRC, bad PID).</td></tr>
<tr><td>2–0</td><td>Reserved.</td></tr>
</table>

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 21-48. 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. 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 21.3.4.3, “USB Interrupt Enable Register (USBINTR),” and Section 21.3.4.2, “USB Status Register (USBSTS).”
4. Set the USBMODE[CM] field to enable device mode, and set the USBMODE[ES] bit for big endian operation.
5. Optionally write the USBCMD register to set the desired interrupt threshold.
6. Set USBMODE[SLOM] to disable setup lockouts.
7. Initialize the EPLISTADDR.
8. 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.”
9. Set the CCM’s UOCSR[BVLD] bit to allow device to connect to a host.
10. 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.

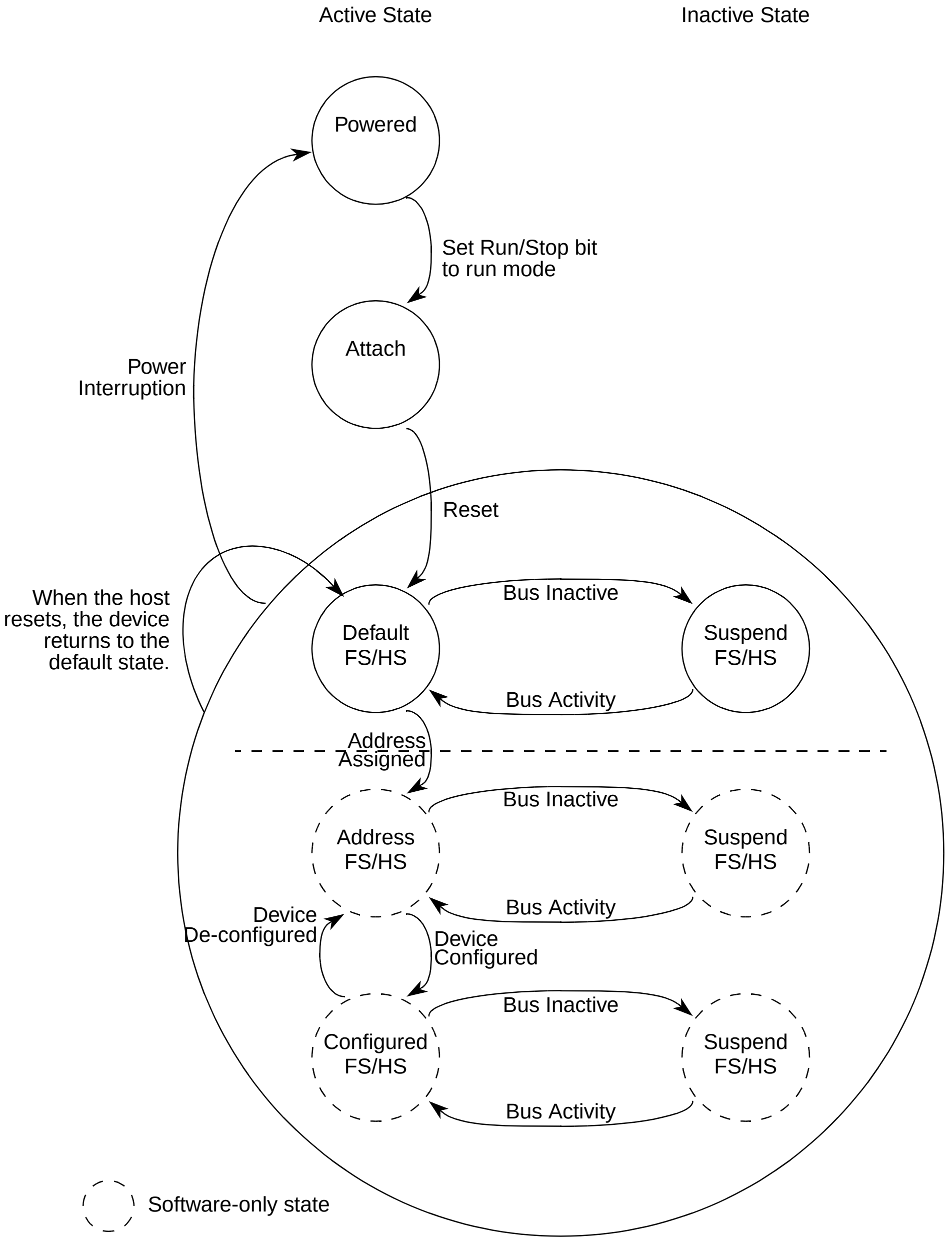


Figure 21-41. USB 2.0 Device States

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 21-49. 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 21-50 shows how to construct a configuration word for endpoint initialization.

Table 21-50. 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 21-51. 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 21-52 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 21-52. 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 21-53. 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 21-54. 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 21-55 shows the device controller response to packets on a control endpoint according to the device controller state.

Table 21-55. 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 21-56. 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 21-56. 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”).

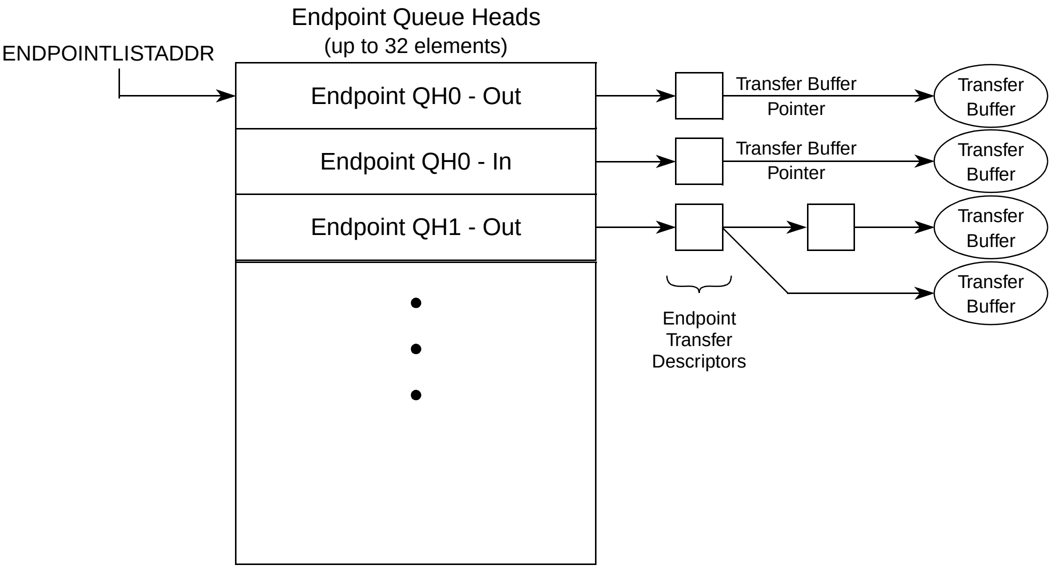


Figure 21-42. Endpoint Queue Head Diagram

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.

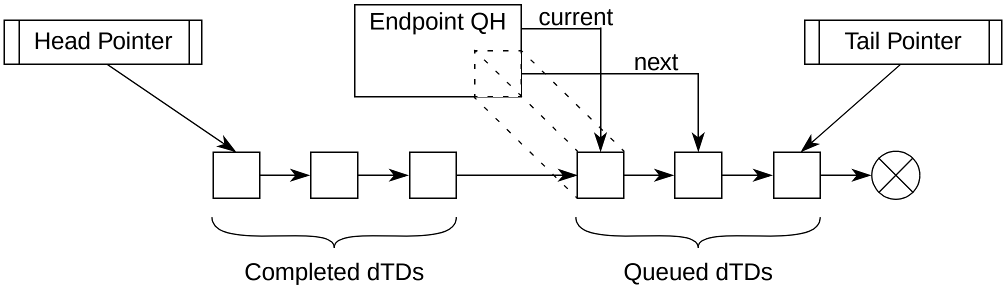


Figure 21-43. Software Link Pointers

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.

CAUTION

> Multiple dTDs can be completed in a single endpoint complete notification. After clearing the notification, the DCD must search the dTD linked list and retire all finished (active bit cleared) dTDs.

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0, Halted = 0, Transaction error = 0, Data buffer error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in Section 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 21-57. 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 21-58. 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 21-59. 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 21-60. 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 21-61. 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 21.3.3.3, “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 21-62.

Table 21-62. 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 PORTSC*n*[PSPD] equals 00 (FS), a LS-pre-PID is sent before transmitting LS traffic.

Maximum packet size must equal 64 or less to prevent undefined behavior.

2. siTD (for direct attach FS) – Periodic (ISO endpoint)

- All FS ISO transactions:
  - Hub address equals 0
  - siTD.EPS equals 00 (full speed)

Maximum packet size must equal to 1023 or less to prevent undefined behavior.

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 21-63 summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

Table 21-63. 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 21.3.4.2, “USB Status Register (USBSTS),” and Section 21.3.4.3, “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 PORTSC*n* 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 PORTSC*n* register for a duration of 10 ms. Due to the complexity required to support the attachment of devices not high speed, a counter is present in the design that can count the 10 ms reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is summarized as:

- Port change interrupt—Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall set the PORTSC*n*[PR] bit to reset the device.
- Software shall clear the PORTSC*n*[PR] bit after 10 ms.
  - This step, necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a 0 to the reset bit while a reset is in progress, the write is ignored and the reset continues until completion.
- Port change interrupt—Port enable change occurs to notify the host controller that the device is now operational and at this point the port speed is determined.

Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation, which re-assigns the port owner for any device that does not connect at high speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner hand-off is not implemented. Therefore, PORTSC*n*[PO] bit is read-only and always reads 0.
- A 2-bit port speed indicator field has been added to PORTSC*n* to provide the current operating speed of the port to the host controller driver.
- A 1-bit high-speed indicator bit has been added to PORTSC*n* to signify that the port is in HS vs. FS/LS.
  - This information is redundant with the 2-bit port speed indicator field above.

Chapter 22
Enhanced Secure Digital Host Controller

22.1 Overview

The enhanced Secure Digital host controller (eSDHC) provides an interface between the host system and several types of memory cards:

- MultiMediaCard (MMC)

  MMC is a universal low-cost data storage and communication medium designed to cover a wide area of applications including mobile video and gaming, which are available from pre-loaded MMC cards or downloadable from cellular phones, WLAN, or other wireless networks. Old MMC cards are based on a seven-pin serial bus with a single data pin, while the new high-speed MMC communication is based on an advanced 11-pin serial bus designed to operate in a low voltage range.
- Secure Digital (SD) card

  The Secure Digital (SD) card is an evolution of MMC technology. It is specifically designed to meet the security, capacity, performance, and environment requirements inherent in the emerging audio and video consumer electronic devices. The physical form factor, pin assignments, and data transfer protocol are forward-compatible with the old MMC.
- SDIO

  Under the SD protocol, the SD cards can be categorized as a memory card, I/O card, or combo card. The memory card invokes a copyright protection mechanism that complies with the security of the SDMI standard. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices. The combo card has both memory and I/O functions. For the sake of simplicity, Figure 22-1 does not show cards with reduced sizes.
- Consumer electronics ATA (CE-ATA)

  CE-ATA uses a hard drive interface optimized for embedded storage applications. The device is layered on top of the MMC protocol stack using the same physical interface. The interface electrical and signaling definition can be found in the MMC specification. Refer to the CE-ATA specification for more details.

The eSDHC acts as a bridge, passing host bus transactions to SD/SDIO/MMC/CE-ATA cards by sending commands and performing data accesses to or from the cards. It handles the SD/SDIO/MMC/CE-ATA protocol at the transmission level. Figure 22-1 shows connection of the eSDHC.

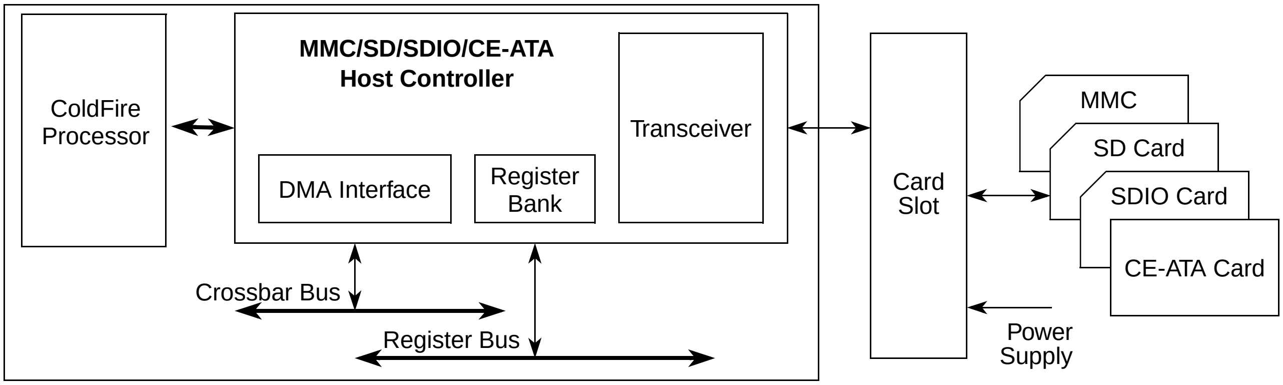


Figure 22-1. System Connection of the eSDHC

22.1.1 Block Diagram

Figure 22-2 is a block diagram of the eSDHC.

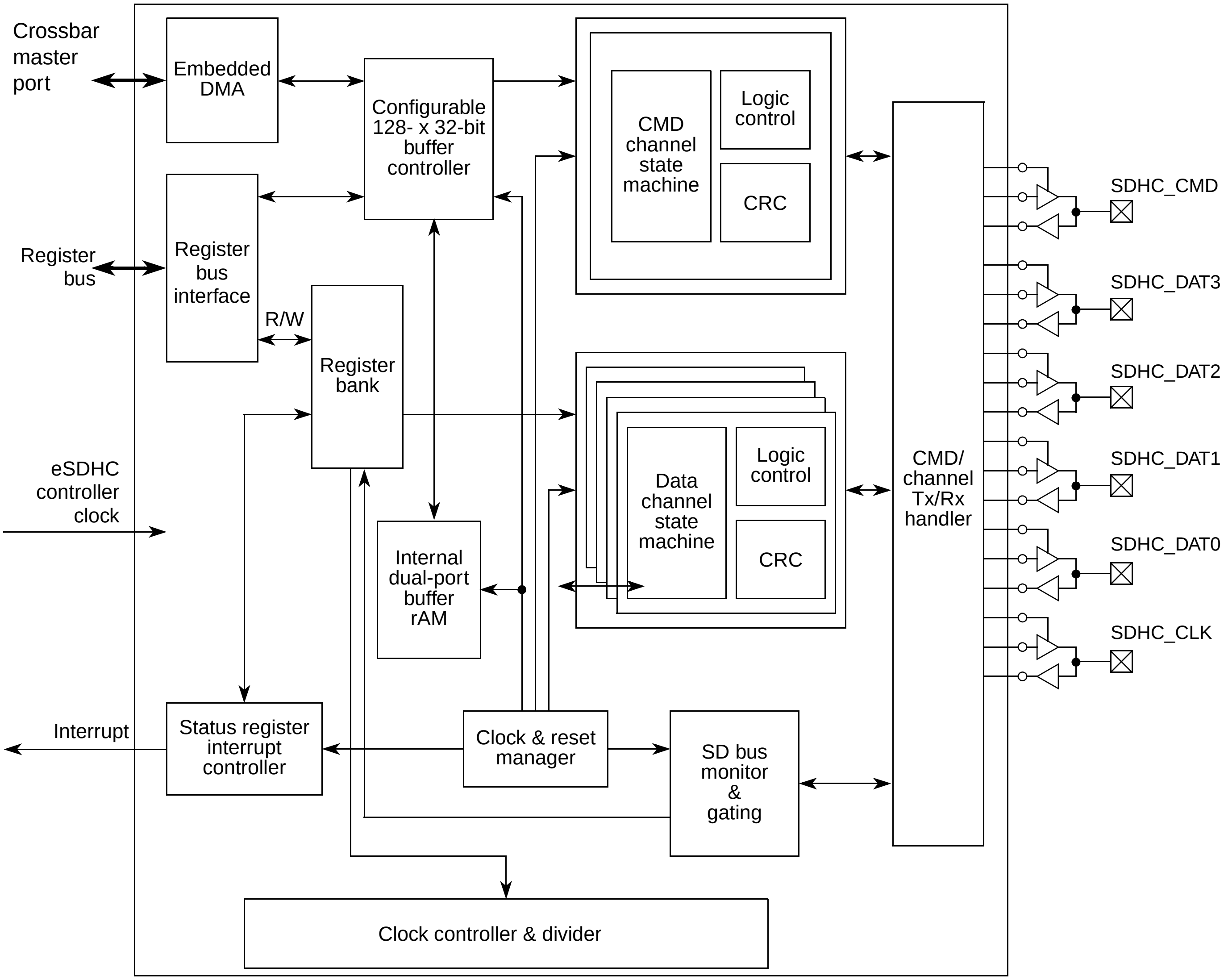


Figure 22-2. eSDHC Block Diagram

22.1.2 Features

The eSDHC includes the following features:

- Compatible with the following specifications:
  - — *SD Host Controller Standard Specification, Version 2.0* (http://www.sdcard.org) with test event register support
  - — *MultiMediaCard System Specification, Version 4.0* (http://www.mmca.org)
  - — *SD Memory Card Specification, Version 2.0* (http://www.sdcard.org), supporting high capacity SD memory cards

- — *SDIO Card Specification, Version 1.2* (http://www.sdcard.org)
- — *CE-ATA Card Specification, Version 1.0* (http://www.sdcard.org)
- Designed to work with CE-ATA, SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMC*plus*, and RS-MMC cards
- SD bus clock frequency up to 25 MHz
- Supports 1-/4-bit SD and SDIO modes, 1-/4-bit MMC modes, 4-/bit CE-ATA devices
  - — Up to 200 Mbps data transfer for SD/SDIO/MMC cards using four parallel data lines
- Single- and multi-block read and write
- Write-protection switch for write operations
- Synchronous and asynchronous abort
- Pause during the data transfer at a block gap
- SDIO read wait and suspend/resume operations
- Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Allows cards to interrupt the host in 1- and 4-bit SDIO modes
- Supports interrupt period, defined in the SDIO standard
- Fully configurable 128 × 32-bit FIFO for read/write data
- Internal DMA capabilities

22.1.3 Data Transfer Modes

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- CE-ATA 1-bit
- CE-ATA 4-bit
- Full-speed mode (up to 25 MHz)

22.2 External Signal Description

The eSDHC contains the following chip I/O signals:

- SDHC_CLK is the internally generated clock signal that drives the CE-ATA, MMC, SD, or SDIO card
- SDHC_CMD I/O sends commands and receive responses from the card
- SDHC_DAT3–SDHC_DAT0 perform data transfers between the eSDHC and the card
  - —

Table 22-1 shows the properties of the eSDHC I/O signals.

Table 22-1. Signal Properties

Name	Port	Function	Reset State	Pull up
SDHC_CLK	O	Clock for MMC/SD/SDIO card	0	N/A
SDHC_CMD	I/O	Command line to card	High Z	Pull up
SDHC_DAT3	I/O	**4-bit mode:** DAT3 line or configured as card detection pin **1-bit mode:** May be configured as card detection pin	High Z	Board should have 100K pull down. The card drives 50K pull up as required by the SD card specification.
SDHC_DAT2	I/O	**4-bit mode:** DAT2 line or read wait **1-bit mode:** Read wait	High Z	Pull up
SDHC_DAT1	I/O	**4-bit mode:** DAT1 line or interrupt detect **1-bit mode:** Interrupt detect	High Z	Pull up
SDHC_DAT0	I/O	DAT0 line or busy-state detect	High Z	Pull up

22.3 Memory Map/Register Definition

Table 22-2 shows the register memory map. All registers can only be accessed based on a 32-bit width.

Table 22-2. eSDHC Memory Map

Address	Register	Width (bits)	Access	Reset	Section/Page
0xFC0C_C000	DMA system address (DSADDR)	32	R/W	0x0000_0000	22.3.1/22-6
0xFC0C_C004	Block attributes (BLKATTR)	32	R/W	0x0001_0000	22.3.2/22-7
0xFC0C_C008	Command argument (CMDARG)	32	R/W	0x0000_0000	22.3.3/22-8
0xFC0C_C00C	Command transfer type (XFERTYP)	32	R/W	0x0000_0000	22.3.4/22-8
0xFC0C_C010	Command response0 (CMDRSP0)	32	R	0x0000_0000	22.3.5/22-11
0xFC0C_C014	Command response1 (CMDRSP1)	32	R	0x0000_0000	22.3.5/22-11
0xFC0C_C018	Command response2 (CMDRSP2)	32	R	0x0000_0000	22.3.5/22-11
0xFC0C_C01C	Command response3 (CMDRSP3)	32	R	0x0000_0000	22.3.5/22-11
0xFC0C_C020	Data buffer access port (DATPORT)	32	R/W	0x0000_0000	22.3.6/22-13
0xFC0C_C024	Present state (PRSSTAT)	32	R	0xFF88_00F8	22.3.7/22-13
0xFC0C_C028	Protocol control (PROCTL)	32	R/W	0x0000_0020	22.3.8/22-17
0xFC0C_C02C	System control (SYSCTL)	32	R/W	0x0000_8008	22.3.9/22-20
0xFC0C_C030	Interrupt status (IRQSTAT)	32	R/W	0x0000_0000	22.3.10/22-22
0xFC0C_C034	Interrupt status enable (IRQSTATEN)	32	R/W	0x117F_013F	22.3.11/22-26
0xFC0C_C038	Interrupt signal enable (IRQSIGEN)	32	R/W	0x0000_0000	22.3.12/22-29
0xFC0C_C03C	Auto CMD12 status (AUTOC12ERR)	32	R	0x0000_0000	22.3.13/22-30
0xFC0C_C040	Host controller capabilities (HOSTCAPBLT)	32	R	0x07E3_0000	22.3.14/22-33
0xFC0C_C044[1]	Watermark level (WML)	32	R/W	0x0810_0810	22.3.15/22-34

Table 22-2. eSDHC Memory Map (continued)

Address	Register	Width (bits)	Access	Reset	Section/Page
0xFC0C_C050	Force event (FEVT)	32	W	0x0000_0000	22.3.16/22-34
0xFC0C_C0FC	Host controller version (HOSTVER)	32	R	0x0000_0001	22.3.17/22-36

[1] The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

22.3.1 DMA System Address Register (DSADDR)

The DMA system address register contains the physical system memory address used for DMA transfers. Only access this register when no transactions are executing (after transactions have stopped). The host driver must wait until PRSSTAT[DLA] is cleared.

Address: 0xFC0C_C000 (DSADDR) Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DS_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 22-3. DMA System Address Register (DSADDR)

Table 22-3. DSADDR Field Descriptions

Field	Description
31–0 DS_ADDR	DMA system address. When the eSDHC stops a DMA transfer, this register points to the system address of the next contiguous data position. **Note:** The DS_ADDR must be aligned to a four-byte boundary; the two least-significant bits must be cleared.

22.3.2 Block Attributes Register (BLKATTR)

The block attributes register configures the number of data blocks and the number of bytes in each block. Only access this register when no transactions are executing (after transactions have stopped). The host driver must wait until PRSSTAT[DLA] is cleared.

Address: 0xFC0C_C004 (BLKATTR)

Access: Read/Write

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

[1] Since at reset XFERTYP[MSBSEL] is cleared, this field reads 0x0001 after reset

Figure 22-4. Block Attributes Register (BLKATTR)

Table 22-4. BLKATTR Field Descriptions

Field	Description
31–16 BLKCNT	Block count for current transfer. This field is enabled when XFERTYP[BCEN] is set and is valid only for multiple block transfers. The host driver should set this field to a value between 1 and the maximum block count. The eSDHC decrements the block count after each block transfer and stops when the count reaches zero. Clearing this field results in no data blocks being transferred. When saving transfer context as a result of a suspend command, this field indicates the number of blocks yet to be transferred. When restoring transfer context prior to issuing a resume command, the host driver should write the previously saved block count. 0000 Stop count 0001 1 block 0002 2 blocks ... FFFF 65,535 blocks **Note:** When XFERTYP[MSBSEL] is cleared, this field always reads 0x0001.
15–13	Reserved
12–0 BLKSIZE	Transfer block size. Specifies the block size for block data transfers. Values can range from one byte up to the maximum buffer size.The DMA always writes at least four bytes to memory. Thus, software should allocate a buffer space rounded up to a 4-byte alighted size in order to avoid data corruption. 0000 No data transfer 0001 1 byte 0002 2 bytes 0003 3 bytes 0004 4 bytes ... 01FF 511 bytes 0200 512 bytes ... 0800 2048 bytes 1000 4096 bytes

22.3.3 Command Argument Register (CMDARG)

The command argument register contains the SD/MMC command argument.

Address: 0xFC0C_C008 (CMDARG) Access: Read/Write

<table>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td colspan="32" rowspan="2">CMDARG</td></tr>
<tr><td>W</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 22-5. Command Argument Register (CMDARG)

Table 22-5. CMDARG Field Descriptions

Field	Description
31–0 CMDARG	Command argument. The SD/MMC command argument is specified as bits 39–8 of the command format in the SD or MMC Specification. If PRSSTAT[CMD] is set, this register is write-protected.

22.3.4 Transfer Type Register (XFERTYP)

The transfer type register controls the operation of data transfers. The host driver should set this register before issuing a command followed by a data transfer, or before issuing a resume command. To prevent data loss, the eSDHC prevents a write to the bits that are involved in the data transfer of this register while the data transfer is active.

The host driver should check PRSSTAT[CDIHB] and PRSSTAT[CIHB] before writing to this register.

- If PRSSTAT[CDIHB] is set, any attempt to send a command with data by writing to this register is ignored.
- If PRSSTAT[CIHB] is set, any write to this register is ignored.

Address: 0xFC0C_C00C (XFERTYP) Access: Read/Write

<table>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td>0</td><td>0</td><td colspan="6" rowspan="2">CMDINX</td><td colspan="2" rowspan="2">CMD TYP</td><td rowspan="2">DP SEL</td><td rowspan="2">CIC EN</td><td rowspan="2">CCC EN</td><td>0</td><td colspan="2" rowspan="2">RSP TYP</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td rowspan="2">MSB SEL</td><td rowspan="2">DTD SEL</td><td>0</td><td rowspan="2">AC1 2EN</td><td rowspan="2">BC EN</td><td rowspan="2">DM AEN</td></tr>
<tr><td>W</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 22-6. Transfer Type Register (XFERTYP)

Table 22-6. XFERTYP Field Descriptions

Field	Description
31–30	Reserved, must be cleared.
29–24 CMDINX	Command index. These bits should be set to the command number (CMD0–63, ACMD0–63) that is specified in bits 45–40 of the command format in the *SD Memory Card Physical Layer Specification* and *SDIO Card Specification*.

Table 22-6. XFERTYP Field Descriptions (continued)

Field	Description
23–22 CMDTYP	Command type. There are three types of special commands: suspend, resume, and abort. Clear this bit field for all other commands. • Suspend command. If the suspend command succeeds, the eSDHC assumes the SD bus has been released and it is possible to issue the next command which uses the SDHC_DAT line. The eSDHC de-asserts read wait for read transactions and stops checking busy for write transactions. In 4-bit mode, the interrupt cycle starts. If the suspend command fails, the eSDHC maintains its current state, and the host driver should restart the transfer by setting PROCTL[CREQ]. The eSDHC does not check if the suspend command succeeds or not. It is the host driver's responsibility to issue a normal CMD52 marked as suspend command when the suspend request is accepted by the card, so that eSDHC can be informed that the SD bus is released and de-assert read wait during read operation. • Resume command. The host driver restarts the data transfer by restoring the registers saved before sending the suspend command and sends the resume command. The eSDHC checks for pending busy state before starting write transfers. • Abort command. If this command is set when executing a read transfer, the eSDHC stops reads to the buffer. If this command is set when executing a write transfer, the eSDHC stops driving the SDHC_DAT line. After issuing the abort command, the host driver should issue a software reset. (Abort transaction) 00 Normal—other commands 01 Suspend—CMD52 for writing bus suspend in the common card control register (CCCR), defined in the SDIO specification 10 Resume—CMD52 for writing function select in CCCR 11 Abort—CMD12, CMD52 for writing I/O abort in CCCR
21 DPSEL	Data present select. Set to indicate that data is present and should be transferred using the SDHC_DAT line. It is cleared for the following: • Commands using only the SDHC_CMD line (e.g. CMD52) • Commands with no data transfer but using busy signal on the SDHC_DAT[0] line (R1b or R5b, e.g. CMD38) **Note:** In resume command, this bit should be set while the other bits in this register should be set the same as when the transfer initially launched. 0 No data present 1 Data present
20 CICEN	Command index check enable. 0 Disable. The index field is not checked. 1 Enable. The eSDHC checks the index field in the response to see if it has the same value as the command index. If it is not, it is reported as a command index error.
19 CCCEN	Command CRC check enable. The number of bits checked by the CRC field value changes according to the length of the response. (Refer to RSPTYP[1:0] and Table 22-8.) 0 Disable. The CRC field is not checked. 1 Enable. The eSDHC checks the CRC field in the response if it contains the CRC field. If an error is detected, it is reported as a command CRC error.
18	Reserved, must be cleared.
17–16 RSPTYP	Response type select. 00 No response 01 Response length 136 10 Response length 48 11 Response length 48 check busy after response
15–6	Reserved, must be cleared.

Table 22-6. XFERTYP Field Descriptions (continued)

Field	Description
5 MSBSEL	Multi/single block select. Enables multiple block SDHC_DAT line data transfers. For any other commands, this bit should be cleared. If this bit is cleared, it is not necessary to set the block count register. (Refer to Table 22-7.) 0 Single block 1 Multiple blocks
4 DTDSEL	Data transfer direction select. Defines the direction of SDHC_DAT line data transfers. The bit is set by the host driver to transfer data from the SD card to the eSDHC and it is cleared for all other commands. 0 Write (host to card) 1 Read (card to host)
3	Reserved, must be cleared.
2 AC12EN	Auto CMD12 enable. Multiple block transfers for memory require CMD12 to stop the transaction. If this bit is set, the eSDHC issues CMD12 automatically when the last block transfer is completed. The host driver should not set this bit to issue commands that do not require CMD12 to stop a multiple block data transfer. In particular, secure commands defined in the Part 3 File Security specification do not require CMD12. In a single block transfer, the eSDHC ignores this bit. 0 Disable 1 Enable
1 BCEN	Block count enable. Enables the block attributes register, which is only relevant for multiple block transfers. When this bit is cleared, the block attributes register is disabled, which is useful in executing an infinite transfer. 0 Disable 1 Enable
0 DMAEN	DMA enable. Enables DMA functionality as described in Section 22.4.2, “DMA Crossbar Switch Interface.” If this bit is set, a DMA operation should begin when the host driver writes to the CMDINX field of the transfer type register. 0 Disable 1 Enable

Table 22-7 shows how register settings determine the data transfer type.

Table 22-7. Determination of Transfer Type

Multi/Single Block Select XFERTYP[MSBSEL]	Block Count Enable XFERTYP[BCEN]	Block Count BLKATTR[BLKCNT]	Function
0	—	—	Single transfer
1	0	—	Infinite transfer
1	1	Non-zero	Multiple transfer
1	1	0	No data transfer

Table 22-8 shows how the response type can be determined by the command index check enable, command CRC check enable, and response type bits.

Table 22-8. Relation Between Parameters and Name of Response Type

Response Type XFERTYP[RSPTYP]	Index Check Enable XFERTYP[CICEN]	CRC Check Enable XFERTYP[CCCEN]	Response Type
00	0	0	No response
01	0	1	R2
10	0	0	R3, R4
10	1	1	R1, R5, R6
11	1	1	R1b, R5b

NOTE

In the SDIO specification, response type notation of R5b is not defined. R5 includes R5b in the SDIO specification. But, R5b is defined in this specification to specify the eSDHC checks busy status after receiving a response. For example, usually CMD52 is used as R5 but the I/O abort command should be used as R5b.

The CRC field for R3 and R4 is expected to be all ones. The CRC check should be disabled for these response types.

22.3.5 Command Response 0–3 (CMDRSP0–3)

The command response registers stores the four parts of the response bits from the card.

Address: 0xFC0C_C010 (CMDRSP0)
0xFC0C_C014 (CMDRSP1)
0xFC0C_C018 (CMDRSP2)
0xFC0C_C01C (CMDRSP3)

Access: Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CMDRSP																															
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. Command Response 0–3 Register (CMDRSP*n*)

Table 22-9 describes the mapping of command responses from the SD bus to the command response registers for each response type. In the table, R[ ] refers to a bit range within the response data as transmitted on the SD bus.

Table 22-9. Response Bit Definition for Each Response Type

Response Type	Meaning of Response	Response Field	Response Register
R1,R1b (normal response)	Card status	R[39:8]	CMDRSP0
R1b (Auto CMD12 response)	Card status for Auto CMD12	R[39:8]	CMDRSP3
R2 (CID, CSD register)	CID/CSD register [127:8]	R[127:8]	{CMDRSP3[23:0], CMDRSP2, CMDRSP1, CMDRSP0}
R3 (OCR register)	OCR register for memory	R[39:8]	CMDRSP0
R4 (OCR register)	OCR register for I/O etc.	R[39:8]	CMDRSP0
R5, R5b	SDIO response	R[39:8]	CMDRSP0
R6 (publish RCA)	New published RCA[31:16] and card status[15:0]	R[39:9]	CMDRSP0

This table shows that:

- Most responses with a length of 48 (R[47:0]) have 32 bits of the response data (R[39:8]) stored in the CMDRSP0 register.
- Responses of type R1b (Auto CMD12 responses) have response data bits R[39:8] stored in the CMDRSP3 register.
- Responses with length 136 (R[135:0]) have 120 bits of the response data (R[127:8]) stored in the CMDRSP0, 1, 2, and 3 registers.

To be able to read the response status efficiently, the eSDHC only stores part of the response data in the command response registers. This enables the host driver to efficiently read 32 bits of response data in one read cycle on a 32-bit bus system. Parts of the response, the index field, and the CRC are checked by the eSDHC (as specified by XFERTYP[CICEN, CCCEN]) and generate an error interrupt if any error is detected. The bit range for the CRC check depends on the response length. If the response length is 48, the eSDHC checks R[47:1], and if the response length is 136, the eSDHC checks R[119:1].

Since the eSDHC may have a multiple block data transfer executing concurrently with a CMD_wo_DAT command, the eSDHC stores the Auto CMD12 response in the CMDRSP3 register and the CMD_wo_DAT response is stored in CMDRSP0. This allows the eSDHC to avoid overwriting the Auto CMD12 response with the CMD_wo_DAT and vice versa. When the eSDHC modifies part of the command response registers it preserves the unmodified bits.

22.3.6 Buffer Data Port Register (DATPORT)

The buffer data port register is a 32-bit data port register used to access the internal buffer.

Address: 0xFC0C_C020 (DATPORT) Access: Read/Write

	31	30	29	28	27	26	25	24	23	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	DATCONT																															
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 22-8. Buffer Data Port Register (DATPORT)

Table 22-10. DATPORT Field Descriptions

Field	Description
31–0 DATCONT	Data content. The buffer data port register is for 32-bit data access by the CPU or an external DMA. When the internal DMA is enabled, any write to this register is ignored, and a read from this register always yields 0.

22.3.7 Present State Register (PRSSTAT)

PRSSTAT indicates the status of the eSDHC to the host driver.

Address: 0xFC0C_C024 (PRSSTAT) Access: Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	DLSL				CLSL	0	0	0	1	0	0	CINS
W																
Reset	1	1	1	1	1	1	1	1	1	0	0	0	1	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	BREN	BWEN	RTA	WTA	SD OFF	PER OFF	HCK OFF	IPG OFF	SDSTB	DLA	CDIHB	CIHB
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0

Figure 22-9. Present State Register (PRSSTAT)

Table 22-11. PRSSTAT Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–28</td><td>Reserved, must be cleared.</td></tr>
<tr><td>27–24
DLSL</td><td>SDHC_DAT[3:0] line signal level. These bits are used to check the SDHC_DAT line level to recover from errors, and for debugging.This is especially useful in detecting the busy signal level from SDHC_DAT[0]. The reset value is affected by the external pull resistors. By default, read value of this bit field after reset is 0111, when SDHC_DAT[3] is pull-down and other lines are pull-up.
<table>
<tr><th>PRSSTAT Bit</th><th>SDHC_DAT<i>n</i></th></tr>
<tr><td>27</td><td>3</td></tr>
<tr><td>26</td><td>2</td></tr>
<tr><td>25</td><td>1</td></tr>
<tr><td>24</td><td>0</td></tr>
</table></td></tr>
<tr><td>23
CLSL</td><td>SDHC_CMD line signal level. This status is used to check the SDHC_CMD line level to recover from errors, and for debugging. The reset value is affected by the external pull resistor, by default, read value of this bit after reset is 1, when the command line is pull-up.</td></tr>
<tr><td>22–17</td><td>Reserved, must be cleared.</td></tr>
<tr><td>16
CINS</td><td>Card inserted. Indicates if a card has been inserted. The eSDHC debounces this signal so that the host driver does not need to wait for it to stabilize. Changing from 0 to 1 generates a card-insertion interrupt in the interrupt status register and changing from 1 to 0 generates a card removal interrupt in the interrupt status register. A write to the force event register does not affect this bit.
The software reset for all in the system control register does not affect this bit. A software reset does not affect this bit.
0 Power-on-reset or no card
1 Card inserted</td></tr>
<tr><td>15–12</td><td>Reserved, must be cleared.</td></tr>
<tr><td>11
BREN</td><td>Buffer read enable. This status is used for non-DMA read transfers. The eSDHC may implement multiple buffers to transfer data efficiently. This read-only flag indicates that a burst-length of valid data exists in the host-side buffer.
When the buffer is read, this bit is cleared. When a burst length of data is ready in the buffer, this bit is set and a buffer read ready interrupt is generated (if the interrupt is enabled).
0 Buffer read disable
1 Buffer read enable</td></tr>
<tr><td>10
BWEN</td><td>Buffer write enable. This status is used for non-DMA write transfers. The eSDHC can implement multiple buffers to transfer data efficiently. This read-only flag indicates if space is available for a burst length of write data.
When the buffer is written, this bit is cleared. When a burst length of data is written to the buffer, this bit is set and a buffer write ready interrupt is generated (if the interrupt is enabled).
0 Buffer write disable
1 Buffer write enable</td></tr>
</table>

Table 22-11. PRSSTAT Field Descriptions (continued)

Field	Description
9 RTA	Read transfer active. This status is used for detecting completion of a read transfer. This bit is set for either of the following conditions: • After the end bit of the read command • When writing a 1 to PROCTL[CREQ] to restart a read transfer This bit is cleared for either of the following conditions: • When the last data block as specified by block length is transferred to the system • When all valid data blocks have been transferred to the system and no current block transfers are being sent as a result of PROCTL[SABGREQ] being set. A transfer complete interrupt is generated when this bit changes to 0. 0 No valid data 1 Transferring data
8 WTA	Write transfer active. This status indicates a write transfer is active. If this bit is 0, it means no valid write data exists in eSDHC. This bit is set in either of the following cases: • After the end bit of the write command. • When writing a 1 to PROCTL[CREQ] to restart a write transfer. This bit is cleared in either of the following cases: • After getting the CRC status of the last data block, as specified by the transfer count (single and multiple) • After getting the CRC status of any block where data transmission is about to be stopped by a stop-at-block-gap request. During a write transaction, a IRQSTAT[BGE] interrupt is generated when this bit is changed to 0, as result of PROCTL[SABGREQ] being set. This status is useful for the host driver in determining when to issue commands during write busy. 0 No valid data 1 Transferring data
7 SDOFF	SDHC clock gated off internally. Indicates the SDHC clock is internally gated off because of a buffer overrun, buffer underrun, a read pause without read-wait assertion, or SYSCTL[SDCLKEN] is cleared to stop the SD clock. This bit is for the host driver to debug data transaction on SD bus.
6 PEROFF	Peripheral clock gated off internally. The host driver uses this bit to debug a transaction on SD bus. When SYSCTL[INITA] is set and the eSDHC is sending 80 clock cycles to the card, SYSCTL[SDCLKEN] must be set to enable the output card clock. Otherwise, the peripheral clock is never gated off.
5 HCKOFF	Crossbar switch master clock internally gated off. The host driver uses this bit to debug a data transfer.
4 IPGOFF	Controller clock gated off internally. Indicates that the controller clock is internally gated off. This bit is for the host driver to debug.
3 SDSTB	SD clock stable. Indicates that the internal card clock is stable. This bit is for the host driver to poll clock status when changing the clock frequency. It is recommended to clear SYSCTL[SDCLKEN] to remove glitches on the card clock when the frequency is changing. 0 Clock is changing frequency and not stable 1 Clock is stable

Table 22-11. PRSSTAT Field Descriptions (continued)

Field	Description
2 DLA	Data line active. Indicates whether one of the SDHC_DAT line on SD bus is in use. For read transactions, this bit indicates if a read transfer is executing on the SD bus. Clearing this bit from 1 to 0 between data blocks generates a block gap event interrupt. This bit is set in either of the following cases: • After the end bit of the read command • When writing a 1 to PROCTL[CREQ] to restart a read transfer This bit is cleared in either of the following cases: • When the end bit of the last data block is sent from the SD bus to the eSDHC • When beginning a read wait transfer initiated by a stop at block gap request The eSDHC waits at the next block gap by driving read wait at the start of the interrupt cycle. If the read-wait signal is already driven (data buffer cannot receive data), the eSDHC can wait for current block gap by continuing to drive the read-wait signal. It is necessary to support read wait in order to use the suspend/resume function. For write transactions, this bit indicates that a write transfer is executing on the SD bus. Clearing this bit from 1 to 0 generates a transfer complete interrupt. This bit is set in any of the following cases: • After the end bit of the write command • When writing a 1 to PROCTL[CREQ] to continue a write transfer This bit is cleared in any of the following cases: • When the SD card releases write-busy of the last data block, the eSDHC also detects if output is not busy. If the SD card does not drive the busy signal after CRC status is received, the eSDHC should consider the card drive not busy. • When the SD card releases write-busy prior to waiting for write transfer as a result of a stop at block gap request 0 SDHC_DAT line inactive 1 SDHC_DAT line active
1 CDIHB	Command inhibit (SDHC_DAT). This bit is set if the SDHC_DAT line is active, the read transfer active is set, or read wait is asserted. If this bit is cleared, it indicates the eSDHC can issue the next SD/MMC command. Commands with busy signal belong to command inhibit (SDHC_DAT) (e.g. R1b and R5b type). Clearing from 1 to 0 generates a transfer complete interrupt. **Note:** The SD host driver can save registers for a suspend transaction after this bit has cleared from 1 to 0. 0 Can issue command which uses the SDHC_DAT line 1 Cannot issue command which uses the SDHC_DAT line
0 CIHB	Command inhibit (SDHC_CMD). This bit is cleared, if the SDHC_CMD line is not in use and the eSDHC can issue a SD/MMC command using the SDHC_CMD line. This bit is set immediately after the XFERTYP register is written. This bit is cleared when the command response is received. Even if the CDIHB bit is set, commands using only the SDHC_CMD line can be issued if this bit is cleared. Clearing from 1 to 0 generates a command complete interrupt. If the eSDHC cannot issue the command because of a command conflict error (refer to command CRC error) or Auto CMD12 error, this bit remains set and IRQSTAT[CC] is not set. 0 Can issue command using only SDHC_CMD line 1 Cannot issue command

NOTE

The host driver can issue CMD0, CMD12, CMD13 (for memory) and CMD52 (for SDIO) when the SDHC_DAT lines are busy during a data transfer. These commands can be issued when PRSSTAT[CIHB] is cleared. Other commands should be issued when PRSSTAT[CDIHB] is cleared. Possible changes to the SD physical specification may add other commands to this list in the future.

22.3.8 Protocol Control Register (PROCTL)

The protocol control register is shown in Figure 22-10.

Address: 0xFC0C_C028 (PROCTL) Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	WE CRM	WE CINS	WE CINT	0	0	0	0	IABG	RW CTL	CREQ	SABG REQ
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	CDSS	CDTL	EMODE		D3CD	DTW		0
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figure 22-10. Protocol Control Register (PROCTL)

Table 22-12. PROCTL Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26 WECRM	Wake-up event enable on SD card removal. This bit enables wakeup event via card removal assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
25 WECINS	Wake-up event enable on SD card insertion. This bit enables wakeup event via card insertion assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
24 WECINT	Wake-up event enable on card interrupt. This bit enables wakeup event via card interrupt assertion in the IRQSTAT register. This bit can be set to 1 if FN_WUS (wake-up support) in CIS is set to 1. 0 Disable 1 Enable
23–20	Reserved, must be cleared.

Table 22-12. PROCTL Field Descriptions (continued)

Field	Description
19 IABG	Interrupt at block gap. This bit is valid only in 4-bit mode of the SDIO card and selects a sample point in the interrupt cycle. If the SDIO card cannot signal an interrupt during a multiple block transfer, this bit should be cleared to avoid an inadvertent interrupt. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. 0 Disable interrupt detection during a multiple block transfer. 1 Enable interrupt detection at the block gap for a multiple block transfer.
18 RWCTL	Read wait control. The read wait function is optional for SDIO cards. If the card supports read wait, set this bit to enable the read wait protocol to stop read data using the SDHC_DAT[2] line. Otherwise, the eSDHC has to stop the SD clock to hold read data, which restricts command generation. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. If the card does not support read wait, this bit should never be set otherwise an SDHC_DAT line conflict may occur. If this bit is cleared, a stop-at-block-gap-during-read operation is also supported, but the eSDHC stops the SD clock to pause the reading operation. 0 Disable read-wait control, and stop SD clock at block gap when the SABGREQ bit is set 1 Enable read-wait control, and assert read wait without stopping the SD clock at block gap when PROCTL[SABGREQ] is set
17 CREQ	Continue request. Restarts a transaction which was stopped using the stop-at-block-gap request. To cancel the request, clear SABGREQ and set this bit to restart the transfer. The eSDHC automatically clears this bit in either of the following cases: • For a read transaction, the PRSSTAT[DLA] bit changes from 0 to 1 as a read transaction restarts. • For a write transaction, the PRSSTAT[WTA] bit changes from 0 to 1 as the write transaction restarts. Therefore, it is not necessary for the host driver to clear. If SABGREQ and this bit are set, the continue request is ignored. 0 No effect 1 Restart
16 SABGREQ	Stop at block gap request. Stops executing a transaction at the next block gap for both DMA and non-DMA transfers. Until the TC bit is set, indicating a transfer completion, the host driver should leave this bit set. Clearing SABGREQ and CREQ does not cause the transaction to restart. Read wait is used to stop the read transaction at the block gap. The eSDHC honors stop-at-block-gap request for write transfers. But for read transfers it requires that the SDIO card support read wait. Therefore, the host driver should not set this bit during read transfers unless the SDIO card supports read wait and has set read wait control to 1. Otherwise, the eSDHC stops the SD bus clock to pause the read operation during the block gap. For write transfers in which the host driver writes data to the data port register, the host driver should set this bit after all block data is written. If this bit is set, the host driver should not write data to the DATPORT register after a block is sent. When this bit is set, the host driver should not clear this bit before IRQSTAT[TC] is set. Otherwise, the eSDHC behavior is undefined. Confirm that IRQSTAT[TC] is enabled. This bit affects PRSSTAT[RTA, WTA, DLA, CIHB]. 0 Transfer 1 Stop or not resume yet
15–8	Reserved, must be cleared.
7 CDSS	Card detect signal selection. Selects the source for card detection. 0 Card detection level is selected (for normal purpose) 1 Card detection test level is selected (for test purpose)
6 CDTL	Card detect test level. Determines card insertion status when CDSS is set. 0 No card in the slot 1 Card is inserted

Table 22-12. PROCTL Field Descriptions (continued)

Field	Description
5–4 EMODE	Endian mode. eSDHC supports only address-invariant mode in data transfer. 00 Reserved 01 Reserved 10 Address-invariant mode. Each byte location in the main memory is mapped to the same byte location in the MMC/SD card. 11 Reserved
3 D3CD	SDHC_DAT3 as card detection pin. If this bit is set, SDHC_DAT3 should be pulled down to act as a card detection pin. Be cautious when using this feature, because SDHC_DAT3 is chip-select for SPI mode, and a pull-down on this pin and CMD0 may set the card into SPI mode, which the eSDHC does not support. **Note:** On this device there is no separate card-detection signal. To use card detection this bit must be set. 0 SDHC_DAT3 does not monitor card insertion 1 SDHC_DAT3 is card0detection pin
2–1 DTW	Data transfer width. Selects the data width of the SD bus. The host driver should set it to match the data width of the card. 00 1-bit mode 01 4-bit mode 10 Reserved 11 Reserved
1	Reserved, must be cleared.

There are three ways to restart the transfer after a stop at the block gap. The appropriate method depends on whether the eSDHC issues a suspend command or the SD card accepts the suspend command:

- If the host driver does not issue a suspend command, the continue request should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card accepts it, a resume command should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card does not accept it, PROCTL[CREQ] should be used to restart the transfer.

Any time PROCTL[SABGREQ] stops the data transfer, the host driver should wait for IRQSTAT[TC] before attempting to restart the transfer. When restarting the data transfer by continue request, the host driver should clear PROCTL[SABGREQ] before or simultaneously.

22.3.9 System Control Register (SYSCTL)

The system control register is shown in Figure 22-11.

Address: 0xFC0C_C02C (SYSCTL)

Access: Read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	INITA	0	0	0	0	0	0	0	DTOCV			
W						RSTD	RSTC	RSTA								
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	SDCLKFS								DVS				SDCLK EN	PEREN	HCKEN	IPGEN
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Figure 22-11. System Control Register (SYSCTL)

Table 22-13. SYSCTL Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27 INITA	Initialization active. When this bit is written ‘1’, 80 SD clocks are sent to the card. After the 80 clocks are sent, this bit is self-cleared. This bit is very useful during the card power-up period when 74 SD clocks are needed and clock auto-gating feature is enabled. Writing one to this bit when it is already set has no effect. Clearing this bit at any time does not affect it. When PRSSTAT[CIHB] or PRSSTAT[CDIHB] is set, writing a one to this bit is ignored. That is, when the command line or data line is active, writing to this bit is not allowed.
26 RSTD	Software reset for SDHC_DAT line. The DMA and part of the data circuit are reset. The following registers and bits are cleared by this bit: • DATPORT register • Buffer is cleared and initialized; PRSSTAT register • PRSSTAT[BREN, BWEN, RTA, WTA, DLA, CDIHB] • PROCTL[CREQ, SABGREQ] • IRQSTAT[BRR, BWR, DINT, BGE, TC] 0 Work 1 Reset
25 RSTC	Software reset for SDHC_CMD line. Only part of the command circuit is reset. The following bits are cleared by this bit: • PRSSTAT[CIHB] • IRQSTAT[CC] 0 Work 1 Reset

Table 22-13. SYSCTL Field Descriptions (continued)

Field	Description
24 RSTA	Software reset for all. This reset affects the entire host controller except for the card-detection circuit. Register bits of type ROC, RW, RW1C, and RWAC are cleared. During its initialization, the host driver should set this bit to reset the eSDHC. The eSDHC should clear this bit when capabilities registers are valid and the host driver can read them. Additional use of the this bit does not affect the value of the capabilities registers. After this bit is set, it is recommended the host driver reset the external card and re-initialize it. 0 Work 1 Reset
23–20	Reserved, must be cleared.
19–16 DTOCV	Data timeout counter value. Determines the interval by which SDHC_DAT line timeouts are detected. Refer to the data timeout error Section 22.3.10, “Interrupt Status Register (IRQSTAT)”, for information on factors that dictate timeout generation. Timeout clock frequency is generated by dividing the base clock SDHC_CLK value by this value. When setting this register, prevent inadvertent timeout events by clearing IRQSTATEN[DTOESEN]. 0000 SDHC_CLK x 2^{13} 0001 SDHC_CLK x 2^{14} ... 1110 SDHC_CLK x 2^{27} 1111 Reserved
15–8 SDCLKFS	SDHC_CLK frequency select. This field, together with DVS, selects the frequency of SDHC_CLK pin. This bit holds the prescaler of the base clock frequency. Only the following settings are allowed: 0x01 Base clock divided by 2 0x02 Base clock divided by 4 0x04 Base clock divided by 8 0x08 Base clock divided by 16 0x10 Base clock divided by 32 0x20 Base clock divided by 64 0x40 Base clock divided by 128 0x80 Base clock divided by 256 Multiple bits must not be set or the behavior of this prescaler is undefined. The maximum SD clock frequency is 25 MHz, and should never exceed this limit. The frequency of SDHC_CLK is set by the following formula: $\text{clock frequency} = (\text{base clock}) / [(\text{SDCLKFS} \times 2) \times (\text{DVS} + 1)]$ *Eqn. 22-1* For example, if the base clock frequency is 96 MHz, and the target frequency is 25 MHz, then choosing the prescaler value of 0x1 and divisor value of 0x1 yields 24 MHz, which is the nearest frequency less than or equal to the target. Similarly, to approach a clock value of 400 KHz, the prescaler value of 0x04 and divisor value of 0xE yields the exact clock value of 400 KHz. The reset value of this bit field is 0x80. So, if the input base clock is about 96 MHz, the default SD clock after reset is 375 KHz.
7–4 DVS	Divisor. Provides a more exact divisor to generate the desired SD clock frequency. The settings are as follows: 0x0 Divide by 1 0x1 Divide by 2 ... 0xE Divide by 15 0xF Divide by 16

Table 22-13. SYSCTL Field Descriptions (continued)

Field	Description
3 SDCLKEN	SDHC_CLK enable. The host controller stops the SDHC_CLK when this bit is cleared. Only change the SDHC_CLK frequency when this bit is cleared. If PRSSTAT[CINS] is cleared, this bit should be cleared by the host driver to save power. 0 SDHC_CLK disabled 1 SDHC_CLK enabled
2 PEREN	Peripheral clock enable. If set, the peripheral clock is always active and no automatic gating is applied, thus SDHC_CLK is active only except auto gating-off during buffer danger. If cleared, the peripheral clock is automatically off when no transaction is on the SD bus. Clearing this bit does not stop SDHC_CLK immediately. The peripheral clock will be internally gated off, if none of the following factors are met: • Command part is reset • Data part is reset • Soft reset • Command is about to send • Clock divisor is just updated • Continue request is just set • This bit is set • Card insertion is detected • Card removal is detected • Card external interrupt is detected • 80 clocks for initialization phase is ongoing 0 The peripheral clock is internally gated off 1 The peripheral clock is not automatically gated off
1 HCKEN	Crossbar switch master clock enable. If set, the clock is always active and no automatic gating is applied. If cleared, the clock is automatically off when no data transfer is on SD bus. 0) Clock is internally gated off 1) Clock is not automatically gated off
0 IPGEN	Controller clock enable. If this bit is set, the controller clock is always active and no automatic gating is applied. The controller clock is internally gated off, if neither the following factors is met: • Command part is reset • Data part is reset • Soft reset • Command is about to send • Clock divisor is just updated • Continue request is just set • This bit is set • Card insertion is detected • Card removal is detected • Card external interrupt is detected • The controller clock is not gated off **Note:** The controller clock is not auto-gated off if the peripheral clock is not gated off. So, clearing this bit only does not take effect if SYSCTL[PEREN] is not cleared. 0 The controller clock is internally gated off 1 The controller clock is not automatically gated off

22.3.10 Interrupt Status Register (IRQSTAT)

An interrupt is generated when one of the status bits and its corresponding interrupt enable bit are set. For all bits, writing one to a bit clears it, while writing zero keeps the bit unchanged. More than one status can be cleared with a single register write. For a card interrupt (IRQSTAT[CINT]), the card must stop asserting the interrupt before writing one to clear. Otherwise, the CINT bit is set again.

Address: 0xFC0C_C030 (IRQSTAT) Access: Read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	DMAE	0	0	0	AC12E	0	DEBE	DCE	DTOE	CIE	CEBE	CCE	CTOE
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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	CINT	CRM	CINS	BRR	BWR	DINT	BGE	TC	CC
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 22-12. Interrupt Status Register (IRQSTAT)

Table 22-14. IRQSTAT Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 DMAE	DMA error. Occurs when internal DMA transfer failed. This bit is set when some error occurs in the data transfer. The value in the DMA system address register is the next fetch address where the error occurs. Since any error corrupts the entire data block, the host driver should restart the transfer from the corrupted block boundary. The address of the block boundary can be calculated from the current DS_ADDR value or the remaining number of blocks and the block size. 0 No Error 1 Error
27–25	Reserved, must be cleared.
24 AC12E	Auto CMD12 error. Occurs when one of the bits in AUTOC12ERR is set. This bit is also set when Auto CMD12 is not executed due to a previous command error. 0 No Error 1 Error
23	Reserved, must be cleared.
22 DEBE	Data end bit error. Occurs when detecting 0 at the end bit position of read data on the SDHC_DAT line or at the end bit position of the CRC. 0 No Error 1 Error **Note:** When DEBE and CINT are set, the software should ignore DEBE. But, it must not ignore the other status bits. The software should also clear this bit by writing 1 to it. It is highly recommended to clear this bit before the next transfer.
21 DCE	Data CRC error. Occurs when detecting CRC error when transferring read data on the SDHC_DAT line or when detecting the write CRC status having a value other than 0b010. 0 No Error 1 Error

Table 22-14. IRQSTAT Field Descriptions (continued)

Field	Description
20 DTOE	Data timeout error. Occurs during one of following timeout conditions: • Busy timeout for R1b and R5b types • Busy timeout after write CRC status • Read data timeout 0 No error 1 Timeout
19 CIE	Command index error. Occurs if a command index error occurs in the command response. 0 No error 1 Timeout
18 CEBE	Command end bit error. Occurs when the end bit of a command response is 0. 0 No error 1 End bit error generated
17 CCE	Command CRC error. A command CRC error is generated in two cases: • If a response is returned and IRQSTAT[CTOE] is cleared (indicating no timeout), this bit is set when detecting a CRC error in the command response. • The eSDHC detects a SDHC_CMD line conflict by monitoring the SDHC_CMD line when a command is issued. If the eSDHC drives the SDHC_CMD line to 1, but detects 0 on the SDHC_CMD line at the next SDHC_CLK edge, then the eSDHC aborts the command (stop driving SDHC_CMD line) and sets this bit. The CTOE bit is also set to distinguish the SDHC_CMD line conflict. 0 No error 1 CRC error generated
16 CTOE	Command timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. Also, if eSDHC detects a SDHC_CMD line conflict, this bit is set along with IRQSTAT[CCE] as shown in Table 22-26. 0 No error 1 Time out
15–9	Reserved, must be cleared.
8 CINT	Card interrupt. • In 1-bit mode, the eSDHC detects the card interrupt without the SD clock to support wakeup. • In 4-bit mode, the card interrupt signal is sampled during the interrupt cycle. So, there are some sample delays between the interrupt signal from the SD card and the interrupt to the host system. Writing 1 clears this bit. But, if the interrupt source from the SD card is not cleared, this bit is set again. To clear this bit, the SD card interrupt source must be cleared followed by writing 1 to this bit. When this bit is set and the host driver needs to start the interrupt service, IRQSIGEN[CINTIEN] should be cleared to stop driving the interrupt signal to the host system. After completing the card interrupt service, write 1 to clear this bit, set IRQSIGEN[CINTIEN], and start sampling the interrupt signal again. 0 No card interrupt 1 Generate card interrupt
7 CRM	Card removal. This bit is set if PRSSTAT[CINS] changes from 1 to 0. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated. When this bit is cleared, it is set again if no card is inserted. To leave it cleared, clear IRQSTATEN[CRMSEN]. 0 Card state unstable or inserted 1 Card removed

Table 22-14. IRQSTAT Field Descriptions (continued)

Field	Description
6 CINS	Card insertion. This bit is set if PRSSTAT[CINS] changes from 0 to 1. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated. When this bit is cleared, it is set again if a card has been inserted. To leave it cleared, clear IRQSTATEN[CINSEN]. 0 Card state unstable or removed 1 Card inserted
5 BRR	Buffer read ready. This bit is set if PRSSTAT[BREN] changes from 0 to 1. 0 Not ready to read buffer 1 Ready to read buffer
4 BWR	Buffer write ready. This bit is set if PRSSTAT[BWEN] changes from 0 to 1. 0 Not ready to write buffer 1 Ready to write buffer
3 DINT	DMA interrupt. Occurs when the internal DMA finishes the data transfer successfully. If errors occur during data transfer, this bit is not set. Instead, the DMAE bit is set. 0 No DMA interrupt 1 DMA interrupt is generated
2 BGE	Block gap event. If PROCTL[SABGREQ] is set, this bit is set when a read or write transaction is stopped at a block gap. If PROCTL[SABGREQ] is cleared, this bit is not set. During a read transaction, this bit is set at the falling edge of the SDHC_DAT line active status (when the transaction is stopped at SD bus timing). Read wait must be supported to use this function. During a write transaction, this bit is set at the falling edge of PRSSTAT[WTA] (after reading the CRC status at SD bus timing). 0 No block gap event 1 Transaction stopped at block gap
1 TC	Transfer complete. This bit is set when a read or write transfer is completed. For a read transaction, this bit is set at the falling edge of PRSSTAT[WTA]. There are two cases in which this interrupt is generated: • When a data transfer is completed, as specified by data length (after the last data has been read to the host system). • When data has stopped at the block gap and completed the data transfer by setting PROCTL[SABGREQ] (after valid data has been read to the host system). For a write transaction, this bit is set at the falling edge of PRSSTAT[DLA]. There are two cases in which this interrupt is generated: • When the last data is written to the SD card, as specified by data length and the busy signal is released. • When data transfers are stopped at the block gap by setting PROCTL[SABGREQ] and data transfers have completed (after valid data is written to the SD card and the busy signal is released).
0 CC	Command complete. This bit is set when the end bit of the command response is received (except Auto CMD12). Refer to PRSSTAT[CIHB]. 0 No command complete 1 Command complete

Table 22-15 below shows that command timeout error has higher priority than command complete. If both bits are set, it can be assumed that the response was not received correctly.

Table 22-15. Relation Between Command Timeout Error and Command Complete Status

Command Complete	Command Timeout Error	Meaning of the Status
0	0	—
Don’t Care	1	Response not received within 64 SDHC_CLK cycles
1	0	Response received

Table 22-16 below shows that transfer complete has higher priority than data timeout error. If both bits are set, the data transfer can be considered complete.

Table 22-16. Relation Between Data Timeout Error and Transfer Complete Status

Transfer Complete	Data Timeout Error	Meaning of the Status
0	0	—
0	1	Timeout occur during transfer
1	X	Data transfer complete

The relation between command CRC error and command timeout error is shown in Table 22-17 below.

Table 22-17. Relation Between Command CRC Error and Command Timeout Error

Command CRC Error	Command Timeout Error	Meaning of the Status
0	0	No error
0	1	Response Timeout Error
1	0	Response CRC Error
1	1	SDHC_CMD line conflict

22.3.11 Interrupt Status Enable Register (IRQSTATEN)

Setting the bits of IRQSTATEN enables the corresponding interrupt status bit to be set by the specified event. If any bit is cleared, the corresponding IRQSTAT bit is also cleared and is never set.

Address: 0xFC0C_C034 (IRQSTATEN) Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	DMAE SEN	0	0	0	AC12E SEN	0	DEBE SEN	DCE SEN	DTOE SEN	CIE SEN	CEBE SEN	CCE SEN	CTOE SEN
W																
Reset	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	CINT SEN	CRM SEN	CINS SEN	BRR SEN	BWR SEN	DINT SEN	BGE SEN	TC SEN	CC SEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1

Figure 22-13. Interrupt Status Enable Register (IRQSTATEN)

Table 22-18. IRQSTATEN Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 DMAESEN	DMA error status enable 0 Masked 1 Enabled
27–25	Reserved, must be cleared.
24 AC12ESEN	Auto CMD12 error status enable 0 Masked 1 Enabled
23	Reserved, must be cleared.
22 DEBESEN	Data end bit error status enable 0 Masked 1 Enabled
21 DCESEN	Data CRC error status enable 0 Masked 1 Enabled
20 DTOESEN	Data timeout error status enable 0 Masked 1 Enabled
19 CIESEN	Command index error status enable 0 Masked 1 Enabled
18 CEBESEN	Command end bit error status enable 0 Masked 1 Enabled
17 CCESEN	Command CRC error status enable 0 Masked 1 Enabled

Table 22-18. IRQSTATEN Field Descriptions (continued)

Field	Description
16 CTOESEN	Command timeout error status enable 0 Masked 1 Enabled
15–9	Reserved, must be cleared.
8 CINTSEN	Card interrupt status enable. If this bit is cleared, the eSDHC clears the interrupt request to the system. The card interrupt detection is stopped when this bit is cleared and restarted when this bit is set. To prevent inadvertent interrupts, the host driver should clear this bit before servicing the card interrupt and should set this bit again after all interrupt requests from the card are cleared. 0 Masked 1 Enabled
7 CRMSEN	Card removal status enable 0 Masked 1 Enabled
6 CINSEN	Card insertion status enable 0 Masked 1 Enabled
5 BRRSEN	Buffer read ready status enable 0 Masked 1 Enabled
4 BWRSEN	Buffer write ready status enable 0 Masked 1 Enabled
3 DINTSEN	DMA interrupt status enable 0 Masked 1 Enabled
2 BGESEN	Block gap event status enable 0 Masked 1 Enabled
1 TCSEN	Transfer complete status enable 0 Masked 1 Enabled
0 CCSEN	Command complete status enable 0 Masked 1 Enabled

NOTE

The eSDHC may sample the card interrupt signal during the interrupt period and hold its value in the flip-flop. As a result of synchronization, there is a delay in the card interrupt (which is asserted from the card) to the time the host system is informed.

To detect a SDHC_CMD line conflict, the host driver must set both CTOESEN and CCESEN bits.

22.3.12 Interrupt Signal Enable Register (IRQSIGEN)

IRQSIGEN selects which interrupt status is indicated to the host system as the interrupt. These status bits all share the same interrupt line. Setting any of these bits enables an interrupt generation. The corresponding status register bit generates an interrupt when the corresponding interrupt signal enable bit is set.

Address: 0xFC0C_C038 (IRQSIGEN) Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	DMAE IEN	0	0	0	AC12E IEN	0	DEBE IEN	DCE IEN	DTOE IEN	CIE IEN	CEBE IEN	CCE IEN	CTOE IEN
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	CINT IEN	CRM IEN	CINS IEN	BRR IEN	BWR IEN	DINT IEN	BGE IEN	TC IEN	CC IEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-14. Interrupt Signal Enable Register (IRQSIGEN)

Table 22-19. IRQSIGEN Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 DMAEIEN	DMA error interrupt enable 0 Masked 1 Enabled
27–25	Reserved, must be cleared.
24 AC12EIEN	Auto CMD12 error interrupt enable 0 Masked 1 Enabled
23	Reserved, must be cleared.
22 DEBEIEN	Data end bit error interrupt enable 0 Masked 1 Enabled
21 DCEIEN	Data CRC error interrupt enable 0 Masked 1 Enabled
20 DTOEIEN	Data timeout error interrupt enable 0 Masked 1 Enabled
19 CIEIEN	Command index error interrupt enable 0 Masked 1 Enabled

Table 22-19. IRQSIGEN Field Descriptions (continued)

Field	Description
18 CEBEIEN	Command end bit error interrupt enable 0 Masked 1 Enabled
17 CCEIEN	Command CRC error interrupt enable 0 Masked 1 Enabled
16 CTOEIEN	Command timeout error interrupt enable 0 Masked 1 Enabled
15–9	Reserved, must be cleared.
8 CINTIEN	Card interrupt signal enable 0 Masked 1 Enabled
7 CRMIEN	Card removal interrupt enable 0 Masked 1 Enabled
6 CINIEN	Card insertion interrupt enable 0 Masked 1 Enabled
5 BRRIEN	Buffer read ready interrupt enable 0 Masked 1 Enabled
4 BWRIEN	Buffer write ready interrupt enable 0 Masked 1 Enabled
3 DINTIEN	DMA interrupt enable 0 Masked 1 Enabled
2 BGEIEN	Block gap event interrupt enable 0 Masked 1 Enabled
1 TCIEN	Transfer complete interrupt enable 0 Masked 1 Enabled
0 CCIEN	Command complete interrupt enable 0 Masked 1 Enabled

22.3.13 Auto CMD12 Error Status Register (AUTOC12ERR)

When IRQSTAT[AC12E] is set, the host driver checks this register to identify what kind of error Auto CMD12 indicated. This register is valid only when IRQSTAT[AC12E] is set.

Address: 0xFC0C_C03C (AUTOC12ERR)

Access: Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	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	CNIB AC12E	0	0	AC12 IE	AC12 CE	AC12 EBE	AC12 TOE	AC12 NE
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 22-15. Auto CMD12 Error Status Register (AUTOC12ERR)

Table 22-20. AUTOC12ERR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 CNIBAC12E	Command not issued by Auto CMD12 error. This bit is set when CMD_wo_DAT is not executed due to an Auto CMD12 error (D04–D01). 0 No error 1 Not Issued
6–5	Reserved, must be cleared.
4 AC12IE	Auto CMD12 index error. Occurs if the command index error occurs in response to a command. 0 No error 1 Error, the CMD index in response is not CMD12
3 AC12CE	Auto CMD12 CRC error. Occurs when detecting a CRC error in the command response. 0 No CRC error 1 CRC error met in Auto CMD12 response
2 AC12EBE	Auto CMD12 end bit error. Occurs when detecting that the end bit of command response is 0 when it should be 1. 0 No error 1 End bit error generated
1 AC12TOE	Auto CMD12 timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. If this bit is set, the other error status bits (2–4) are meaningless. 0 No error 1 Time out
0 AC12NE	Auto CMD12 not executed. If a memory multiple block data transfer is not started due to command error, this bit is not set because it is not necessary to issue Auto CMD12. Setting this bit means eSDHC cannot issue Auto CMD12 to stop the memory multiple block data transfer due to some error. If this bit is set, the other error status bits (1–4) are meaningless. 0 Executed 1 Not executed

Table 22-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12

Auto CMD12 CRC Error	Auto CMD12 Timeout Error	Types of Error
0	0	No error

Table 22-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12 (continued)

Auto CMD12 CRC Error	Auto CMD12 Timeout Error	Types of Error
0	1	Response timeout error
1	0	Response CRC error
1	1	SDHC_CMD line conflict

There are three scenarios when AUTOC12ERR can be changed:

1. When eSDHC is going to issue Auto CMD12
   - Set AC12NE if Auto CMD12 cannot be issued due to an error in the previous command.
   - Clear AC12NE if Auto CMD12 is issued.
2. At the end bit of an Auto CMD12 response
   - Check received responses by checking the error bits 1–4.
   - Set if error is detected.
   - Clear if error is not detected.
3. Before reading AUTOC12ERR[CNIBAC12E]
   - Set CNIBAC12E if there is a command that cannot be issued
   - Clear CNIBAC12E if there is no command to issue

The timing of generating the Auto CMD12 error and writing to the command register is asynchronous. The command may be blocked by any Auto CMD12 error causing CNIBAC12E to be set. Therefore, it is suggested to read this register only when IRQSTAT[AC12E] is set. An Auto CMD12 error interrupt is generated when one of the error bits 0–4 is set1. The CNIBAC12E error bit does not generate an interrupt.

22.3.14 Host Controller Capabilities (HOSTCAPBLT)

The host controller capabilities provides the host driver with information specific to the eSDHC implementation. The value in this register does not change in a software reset, and any write to this register is ignored.

Address: 0xFC0C_C040 (HOSTCAPBLT) Access: Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	VS18	VS30	VS33	SRS	DMAS	HSS	0	0	MBL		
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	0	0	0	1	1

	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 22-16. Host Capabilities Register (HOSTCAPBLT)

Table 22-22. HOSTCAPBLT Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26 VS18	Voltage support 1.8 V. This bit depends on the host system ability. 0 1.8 V not supported 1 1.8 V supported
25 VS30	Voltage support 3.0 V. This bit depends on the host system ability. 0 3.0 V not supported 1 3.0 V supported
24 VS33	Voltage support 3.3 V. This bit depends on the host system ability. 0 3.3 V not supported 1 3.3 V supported
23 SRS	Suspend/resume support. Indicates if eSDHC supports suspend/resume functionality. If this bit is 0, the suspend and resume mechanism, as well as the read wait, are not supported and the host driver should not issue suspend or resume commands. 0 Not supported 1 Supported
22 DMAS	DMA support. Indicates if eSDHC is capable of using internal DMA to transfer data between system memory and the data buffer directly. 0 DMA not supported 1 DMA supported
21 HSS	High speed support. Indicates if the eSDHC supports high speed mode and the host system can supply the SD clock frequency from 25 to 50 MHz. 0 High speed not supported 1 High speed supported
20–19	Reserved, must be cleared.

Table 22-22. HOSTCAPBLT Field Descriptions (continued)

Field	Description
18–16 MBL	Max block length. Indicates the maximum block size that the host driver can read and write to the buffer in the eSDHC. The buffer should transfer block size without wait cycles. 000 512 bytes 001 1024 bytes 010 2048 bytes 011 4096 bytes
15–0	Reserved, must be cleared.

22.3.15 Watermark Level Register (WML)

Both write and read watermark levels are configurable. The value can be any number from 1–128 words.

Address: 0xFC0C_C044 (WML) Access: Read/Write

	31	30	29	28	27	26	25	24	23	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	1	0	0	0	WR_WML								0	0	0	0	1	0	0	0	RD_WML							
W																																
Reset	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0

Figure 22-17. Watermark Level Register (WML)

Table 22-23. WML Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23–16 WR_WML	Write watermark level. Number of 32-bit words of watermark level in DMA write operation. Also, the number of words of write burst length.
15–8	Reserved, must be cleared.
7–0 RD_WML	Read watermark level. Number of 32-bit words of watermark level in DMA read operation. Also, the number of words of read burst length. **Note:** The maximum value for RD_WML is 0x10, which means 16 words (64 bytes). Setting RD_WML to a higher value results in non-predicted behavior.

22.3.16 Force Event Register (FEVT)

The force event register is not a physically implemented register. Rather, it is an address to which the IRQSTAT register can be written if the corresponding bit of IRQSTATEN is set. Therefore, this register is a write-only register and writing zero has no effect. Writing 1 to this register sets the corresponding bit of IRQSTAT. Reading from this register always returns zeroes.

Forcing a card interrupt generates a short pulse on the SDHC_DAT[1] line, and the driver may treat this interrupt as normal. The interrupt service routine may skip polling the card-interrupt source as the interrupt is self-cleared.

Address: 0xFC0C_C050 (FEVT)

Access: 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	FEVT CINT			FEVT DMAE				FEVT AC12E		FEVT DEBE	FEVT DCE	FEVTD TOE	FEVT CIE	FEVT CEBE	FEVT CCE	FEVT CTOE
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									FEVTCNI BAC12E			FEVT AC12IE	FEVTA C12EBE	FEVTA C12CE	FEVTA C12TOE	FEVTA C12NE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-18. Force Event Register (FEVT)

Table 22-24. FEVT Field Descriptions

Field	Description
31 FEVTCINT	Force event card interrupt. Writing 1 to this bit generates a low-level short pulse on the internal SDHC_DAT[1] line, which imitates a self-clearing interrupt from the external card. If enabled, IRQSTAT[CINT] is set and the interrupt service routine may treat this interrupt as a normal interrupt from the external card.
30–29	Reserved, must be cleared.
28 FEVTDMAE	Force event DMA error. Forces IRQSTAT[DMAE] to set.
27–25	Reserved, must be cleared.
24 FEVTAC12E	Force event Auto CMD12 error. Forces IRQSTAT[AC12E] to set.
23	Reserved, must be cleared.
22 FEVTDEBE	Force event data end bit error. Forces IRQSTAT[DEBE] to set.
21 FEVTDCE	Force event data CRC error. Forces IRQSTAT[DCE] to set.
20 FEVTDTOE	Force event data time out error. Forces IRQSTAT[DTOE] to set.
19 FEVTCIE	Force event command index error. Forces IRQSTAT[CCE] to set.
18 FEVTCEBE	Force event command end bit error. Forces IRQSTAT[CEBE] to set.
17 FEVTCCE	Force event command CRC error. Forces IRQSTAT[CCE] to set.
16 FEVTCCE	Force event command time out error. Forces IRQSTAT[CTOE] to set.

Table 22-24. FEVT Field Descriptions (continued)

Field	Description
15–8	Reserved, must be cleared.
7 FEVTCNIBAC12E	Force event command not executed by Auto CMD12 error. Forces AUTOC12ERR[CNIBAC12E] to set.
6–5	Reserved, must be cleared.
4 FEVTAC12IE	Force event Auto CMD12 index error. Forces AUTOC12ERR[AC12IE] to set.
3 FEVTAC12EBE	Force event Auto CMD12 end bit error. Forces AUTOC12ERR[AC12EBE] to set.
2 FEVTAC12CE	Force event Auto CMD12 CRC error. Forces AUTOC12ERR[AC12CE] to set.
1 FEVTAC12TOE	Force event Auto CMD12 time out error. Forces AUTOC12ERR[AC12TOE] to set.
0 FEVTAC12NE	Force event Auto CMD12 not executed. Forces AUTOC12ERR[AC12NE] to set.

22.3.17 Host Controller Version Register (HOSTVER)

The host controller version register contains the version for the vendor and the host controller. All the bits are read-only.

Address: 0xFC0C_C0FC (HOSTVER)

Access: Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	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	VVN								SVN							
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 22-19. Host Controller Version Register (HOSTVER)

Table 22-25. HOSTVER Field Descriptions

Field	Description
31–16	Reserved, must be cleared.

Table 22-25. HOSTVER Field Descriptions (continued)

Field	Description
15–8 VVN	Vendor version number. The host driver should not use this status. The upper and the lower 4-bits indicate the version. 0x00 Freescale eSDHC version 1.0 0x10 Freescale eSDHC version 2.0 0x11 Freescale eSDHC version 2.1 0x12 Freescale eSDHC version 2.2 others Reserved
7–0 SVN	Specification version number. Indicates for the host controller specification version. The upper and the lower 4-bits indicate the version. 0x00 SD Host Specification Version 1.0 0x01 SD Host Specification Version 2.0, supports the test event register. others Reserved

22.4 Functional Description

The following sections provide a brief functional description of the major system blocks, including the data buffer, DMA crossbar switch interface, register bank, register bus interface, dual-port memory wrapper, data/command controller, clock and reset manager, and clock generator.

22.4.1 Data Buffer

The eSDHC uses one configurable data buffer so that data can be transferred between the internal system bus and the SD card in an optimized manner to maximize throughput between the two clock domains (the peripheral clock and the crossbar switch master clock). See Figure 22-20 for an illustration of the buffer scheme.

The buffer is used as temporary storage for data being transferred between the host system and the card. The burst lengths for read and write are both configurable and can be any value between 1 and 128 words.

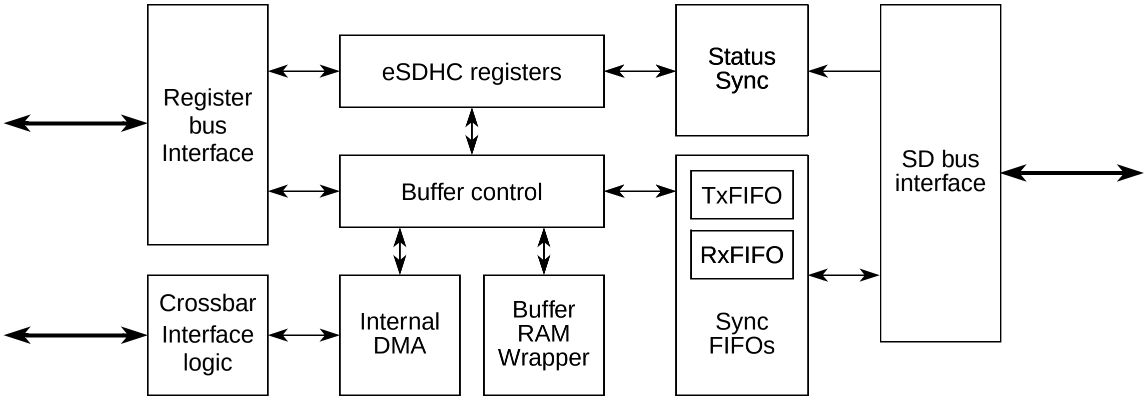


Figure 22-20. eSDHC Buffer Scheme

For a host read operation, when the amount of data exceeds the RD_WML value, the eSDHC sets PRSSTAT[BREN] and either:

- Issues a DMA request to inform the system to read the data

- Issues a DMA interrupt to inform the system to read the data
- When granted crossbar access permission, the internal DMA burst-reads RD_WML number of words

Conversely, for a host write operation, when the amount of buffer spaces exceeds the WR_WML value, the eSDHC sets PRSSTAT[BWEN] and either:

- Issues a DMA request to inform the system to write data to the buffer
- Issues a DMA interrupt to inform the system to write data to the buffer
- When granted crossbar access permission, the internal DMA burst-writess WR_WML number of words into the buffer

22.4.1.1 Write Operation Sequence

There are three ways to write data into the buffer when the user transfers data to the card.

- The external DMA through the eSDHC DMA request signal
- The processor core polling IRQSTAT[BWR] (interrupt or polling)
- The internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts an external DMA request when more than WML[WR_WML] number of empty buffer word slots are available and ready for receiving new data. At the same time, the eSDHC sets IRQSTAT[BWR]. The buffer write ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferredand no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the write operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[WR_WML] number of words of data can be held in the buffer. If the buffer is empty and the host system does not write data in time, the eSDHC stops the SDHC_CLK to avoid a data buffer underrun situation.

22.4.1.2 Read Operation Sequence

There are three ways to read data from the buffer when transferring data to the card.

- The external DMA through the eSDHC DMA request signal
- The processor core polling IRQSTAT[BRR] (interrupt or polling)
- The internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts a DMA request when more than WML[RD_WML] number of words are available and ready for the system to fetch the data. At the same time, the eSDHC sets the IRQSTAT[BRR] bit. The buffer read ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the read operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[RD_WML] number of words of data are in the buffer. If the buffer is full and the host system does not read the data in time, the eSDHC stops the SDHC_CLK to avoid a data buffer overrun situation.

22.4.1.3 Data Buffer Size

To use the buffer in the most optimized way, the buffer size must be known. In the eSDHC the data buffer can hold up to 128 32-bit words, and the read and write watermark levels are each configurable from 1–128 words. The host driver may configure the values according to the system situation and requirements.

During multi-block data transfer, the maximum block length is 4096 bytes, which can satisfy all the requirements from CE-ATA, MMC, SD, and SDIO cards. Any block length less than this value is also allowed. The only restriction is from the external card since it may not support such a large block or a partial block access that is not an integer multiple of 512 bytes.

22.4.1.4 Dividing Large Data Transfer

This SDIO command CMD53 definition limits the maximum data size of data transfers according to the following formula:

$$\text{Maximum data size} = (\text{block size}) \times (\text{block count}) \qquad \textit{Eqn. 22-2}$$

The length of a multiple block transfer must be in block size units. If the total data length cannot be divided evenly to a multiple of the block size, then there are two ways to transfer the data depending on the function and card design.

- The card driver splits the transaction. The remainder of block size data is then transferred using a single block command at the end.
- Add dummy data in the last block to fill the block size. The card must remove the dummy data.

See Figure 22-21 for an example of dividing large data transfers. Although the eSDHC supports a block size of up to 4096 bytes, the example below illustrates a maximum of 64 bytes where the data must be divided.

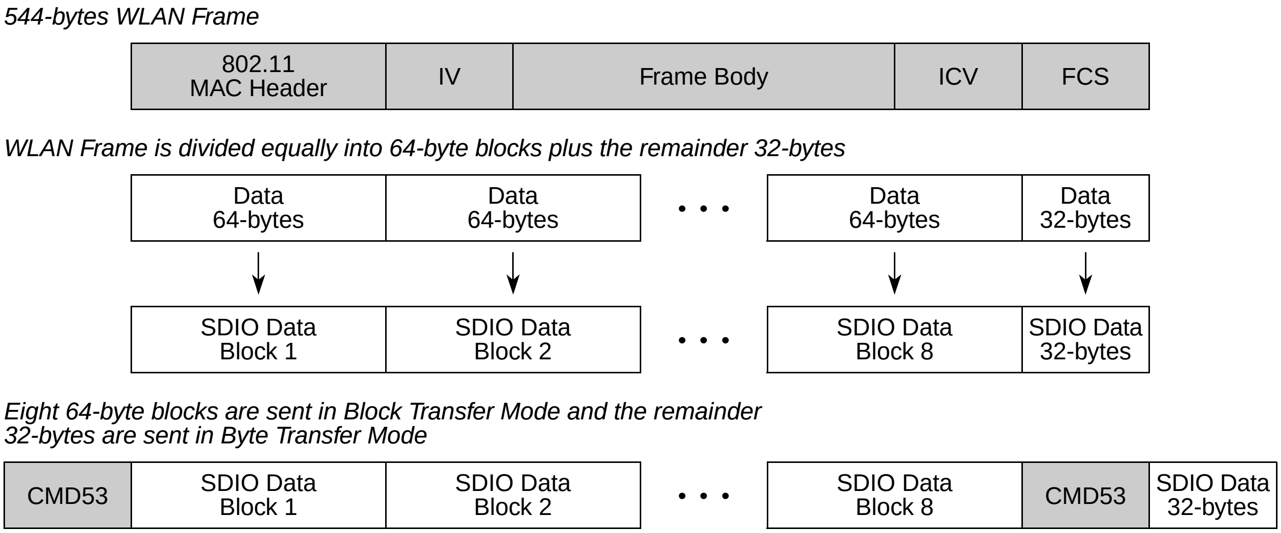


Figure 22-21. Example of Dividing a Large Data Transfer

22.4.1.5 External DMA Request

When the internal DMA is not enabled, the data buffer generates a DMA request to the system. During a write operation, when the number of WR_WML words can be held in the buffer, a DMA request is asserted to the processor for a DMA burst write. IRQSTAT[BWR] is also set if IRQSTATEN[BWRSEN] is set. The DMA request is immediately deasserted when an access on the data port register is made. If another write burst is allowed, the DMA request is asserted again after a cycle.

Likewise, during a read operation, when the number of RD_WML words are in the buffer, a DMA request is asserted to the processor for a DMA burst read. IRQSTAT[BRR] is also set if IRQSTATEN[BRRSEN] is set. The DMA request is immediately deasserted when an access on the data port register is made. If another read burst is allowed, the DMA request is asserted again after a cycle.

Since the DMA burst length cannot change during a data transfer, the read or write burst length must be a divisor of the block size. For example, if the block size is 512 bytes, the burst length must be 1, 2, 4, 8, 16..., or 128 words.

22.4.2 DMA Crossbar Switch Interface

The internal DMA implements a DMA engine and crossbar switch master. When the internal DMA is enabled (XFERTYP[DMAEN] is set), the buffer interrupt status bits are still set if they are enabled. To avoid setting them, clear IRQSTATEN[BWRSEN, BRRSEN]. See Figure 22-22 for illustration of the DMA crossbar switch interface block.

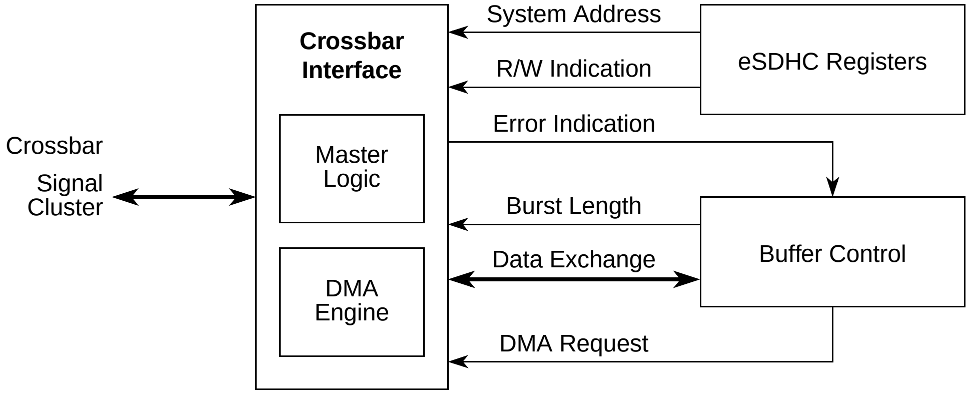


Figure 22-22. DMA Crossbar Switch Interface Block

22.4.2.1 Internal DMA Request

If the watermark level is met in the data transfer and the internal DMA is enabled, the data buffer block sends a DMA request to this block. Meanwhile, the external DMA request is disabled. The delay of response from the internal DMA engine depends on the system bus loading and the priority assigned to eSDHC. The DMA engine does not respond to the request during its burst transfer, and is available as soon as the burst is over. The data buffer deasserts the request once an access on the buffer is made. Upon access to the buffer by the internal DMA, the data buffer updates its internal buffer pointer and when the watermark level is satisfied, another DMA request is sent.

The data transfer is in the block unit and the last watermark level is always set to the remaining number of words. For instance, for a multi-block data read with each block size of 31 bytes, the burst length is set at six words. After the first burst transfer, if there are more than seven bytes in the buffer, which might be partly some data of the next block, another DMA read request is sent because the remaining number of words to send for the current block is (31 – 6 × 4) ÷ 4 = 2, and eSDHC reads two words out of the buffer, with seven valid bytes and one stuff byte automatically added by eSDHC.

22.4.2.2 DMA Burst Length

Just like the CPU polling access, the DMA burst length for the internal DMA engine does not a restriction other than the maximum size. The burst length for read or write can be 1–128 words. The actual burst length for the DMA depends on which is smaller: configured watermark level or the remaining words of current block.

Take the example in Section 22.4.2.1, “Internal DMA Request,” again. After six words are read, the burst length is two words to complete the 31-byte block. The burst length then changes back to six words to prepare for the next 31-byte block. The host driver writer may take this variable burst length into account. It is also acceptable to configure the burst length as the divisor of block size so that each time the burst length is the same.

22.4.2.3 Crossbar Switch Master Interface

It is possible that the internal DMA engine fails during the data transfer. When an error occurs, the DMA engine stops the transfer and goes to the idle state, while the internal data buffer stops working, too. IRQSTAT[DMAE] is set to inform the driver.

Once the IRQSTAT[DMAE] interrupt is received, software should send CMD12 to abort the current transfer and read DSADDR[DS_ADDR] to obtain the start address of the corrupted block. After the DMA error is fixed, the software should apply a data reset and restart the transfer from this address to recover the corrupted block.

22.4.3 SD Protocol Unit

The SD protocol unit deals with all SD protocol affairs and performs the following:

- Acts as the bridge between internal buffer and the SD bus
- Sends the command data and its argument serially
- Stores the serial response bit stream into corresponding registers
- Detects bus state on SDHC_DAT[0] line
- Monitors interrupt from the SDIO card
- Asserts read wait signal
- Gates off SD clock when the buffer announces danger status
- Detects write-protect state
- And other functions

It consists of four submodules: SD transceiver, SD clock and monitor, command agent and data agent.

22.4.3.1 SD Transceiver

In the SD protocol unit, the transceiver is the main control module. It consists of a FSM and the control module, from which the control signals for all other three modules are generated.

22.4.3.2 SD Clock & Monitor

This module monitors the signal level on all four data lines and the command lines, directly route the level values into the register bank for the driver to debug with.

The transceiver reports the card insertion state according to the signal level on SDHC_DAT[3] line when PROCTL[D3CD] is set.

If the internal data buffer is in danger and the SD clock must be gated off to avoid buffer over/underrun, this module asserts the gate of output SD clock to shut the clock off. When the buffer danger is eliminated when system access of the buffer catches up, the clock gate of this module is open and the SD clock is active again.

22.4.3.3 Command Agent

The command agent deals with the transactions on SDHC_CMD line. See Figure 22-23 for illustration of the structure for the command CRC shift register.

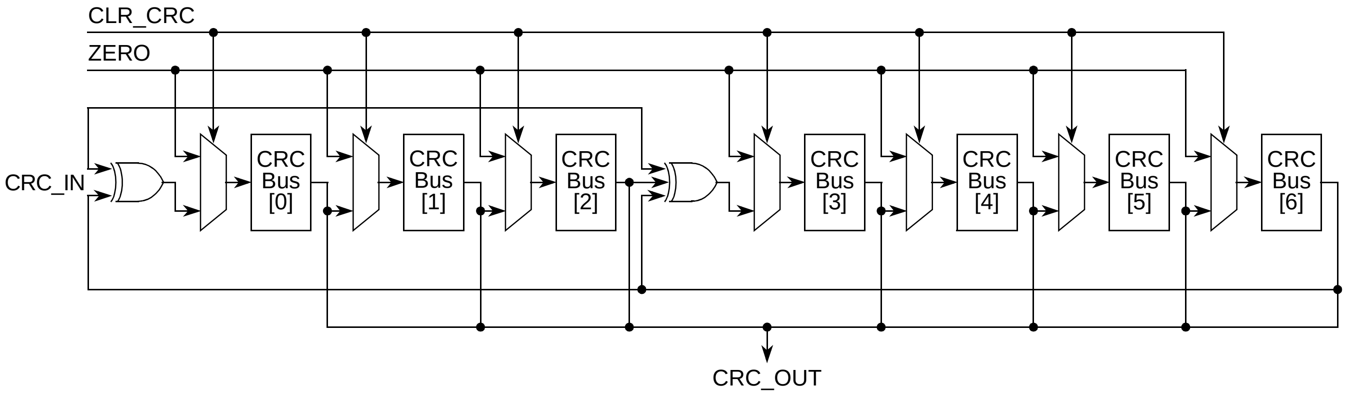


Figure 22-23. Command CRC Shift Register

The CRC polynomials for the SDHC_CMD are as follows:

Generator polynomial: $G(x) = x^7 + x^3 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + ... + (\text{last bit}) \times x^0$

$CRC[6:0] = \text{Remainder } [(M(x) \times x^7) \div G(x)]$

22.4.3.4 Data Agent

The data agent handles the transactions on the eight data lines. Moreover, this module also detects the busy state from on SDHC_DAT[0] line, and generates read wait state by the request from the transceiver. The CRC polynomials for the SDHC_DAT are as follows:

Generator polynomial: $G(x) = x^{16} + x^{12} + x^5 + 1$

$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + ... + (\text{last bit}) \times x^0$

$CRC[15:0] = \text{Remainder } [(M(x) \times x^{16}) \div G(x)]$

22.4.4 Clock & Reset Manager

This module controls all the reset signals within the eSDHC. There are four types of reset signals within eSDHC: hardware reset, software reset for all, software reset for data, and software reset for command. All these signals are fed into this module and stable signals are generated to reset all other modules.

This module also gates off all the inside signals. The module monitors the activities of all other modules, supplies the clocks for them, and when enabled, automatically gates off the corresponding clocks.

22.4.5 Clock Generator

The clock generator generates the SDHC_CLK by dividing the internal bus clock into two stages. Refer to Figure 22-24 for the structure of the divider, in which the term base represents the frequency of the internal bus clock.

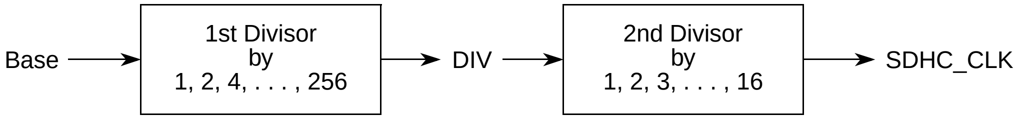


Figure 22-24. Two Stages of the Clock Divider

The first stage is a prescaler. The frequency of clock output from this stage, DIV, can be base, base/2, base/4, ..., or base/256.

The second stage outputs the actual clock, SDHC_CLK, as the driving clock for all sub-modules of SD protocol unit, and the sync FIFOs in Figure 22-20 to synchronize with the data rate from the internal data buffer. It can be div, div/2, div/3,..., or div/16. Thus, the highest frequency of SDHC_CLK generated by the internal bus clock is base, while the lowest frequency is base/4096.

22.4.6 SDIO Card Interrupt

22.4.6.1 Interrupts in 1-bit Mode

In this case the SDHC_DAT[1] pin is dedicated to providing the interrupt function. An interrupt is asserted by pulling the SDHC_DAT[1] low from the SDIO card, until the interrupt service is finished to clear the interrupt.

22.4.6.2 Interrupt in 4-bit Mode

Since the interrupt and data line 1 share pin 8 in four-bit mode, an interrupt is only sent by the card and recognized by the host during a specific time. This is known as the interrupt period. The eSDHC only samples the level on pin 8 during the interrupt period. At all other times, the host ignores the level on pin 8 and treats it as the data signal. The definition of the interrupt period is different for operations with single- and multiple-block data transfers.

For normal single data block transmissions, the interrupt period becomes active two clock cycles after the completion of a data packet. This interrupt period lasts until after the card receives the end bit of the next command that has a data block transfer associated with it.

For multiple block data transfers in 4-bit mode there is only a limited period of time that the interrupt period can be active due to the limited period of data line availability between the multiple blocks of data. This requires a more strict definition of the interrupt period. For this case, the interrupt period is limited to two clock cycles, which begins two clocks after the end bit of the previous data block. During this two-clock cycle interrupt period if an interrupt is pending, the SDHC_DAT[1] line is held low for one clock cycle with the last clock cycle pulling SDHC_DAT[1] high. On completion of the interrupt period, the card releases the SDHC_DAT[1] line into the high-Z state. The eSDHC samples the SDHC_DAT[1] during the interrupt period when PROCTL[IABG] is set.

Refer to *SDIO Card Specification v1.10f* for further information about the SDIO card interrupt.

22.4.6.3 Card Interrupt Handling

When IRQSIGEN[CINTIEN] is cleared, the eSDHC clears the interrupt request to the host system. The host driver should clear this bit before servicing the SDIO interrupt and should set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

If enabled by IRQSTATEN[CINTSEN], the IRQSTAT[CINT] bit can only be cleared by resetting the SDIO interrupt source and then writing one to this bit. Merely writing to this bit has no effect.

In 1-bit mode, the eSDHC detects the SDIO interrupt with or without SD clock (to support wakeup). In 4-bit mode, the interrupt signal is sampled during the interrupt period, so there are some sample delays between the interrupt signal from the SDIO card and the interrupt to the host system interrupt controller. When IRQSTAT[CINT] is set and the host driver needs to start this interrupt service, IRQSTATEN[CINTSEN] is cleared in order to clear IRQSTAT[CINT] that is latched in the eSDHC and to stop driving the interrupt signal to the processor’s interrupt controller. The host driver must issue a CMD52 to clear the interrupts at the card. After completion of the card interrupt service, IRQSTATEN[CINTSEN] is set, and the eSDHC can start sampling the interrupt signal again.

See the following illustrations:

- Figure 22-25 (a) for an illustration of the SDIO card interrupt scheme
- Figure 22-25 (b) for the sequences of software and hardware events that take place during card interrupt handling procedure

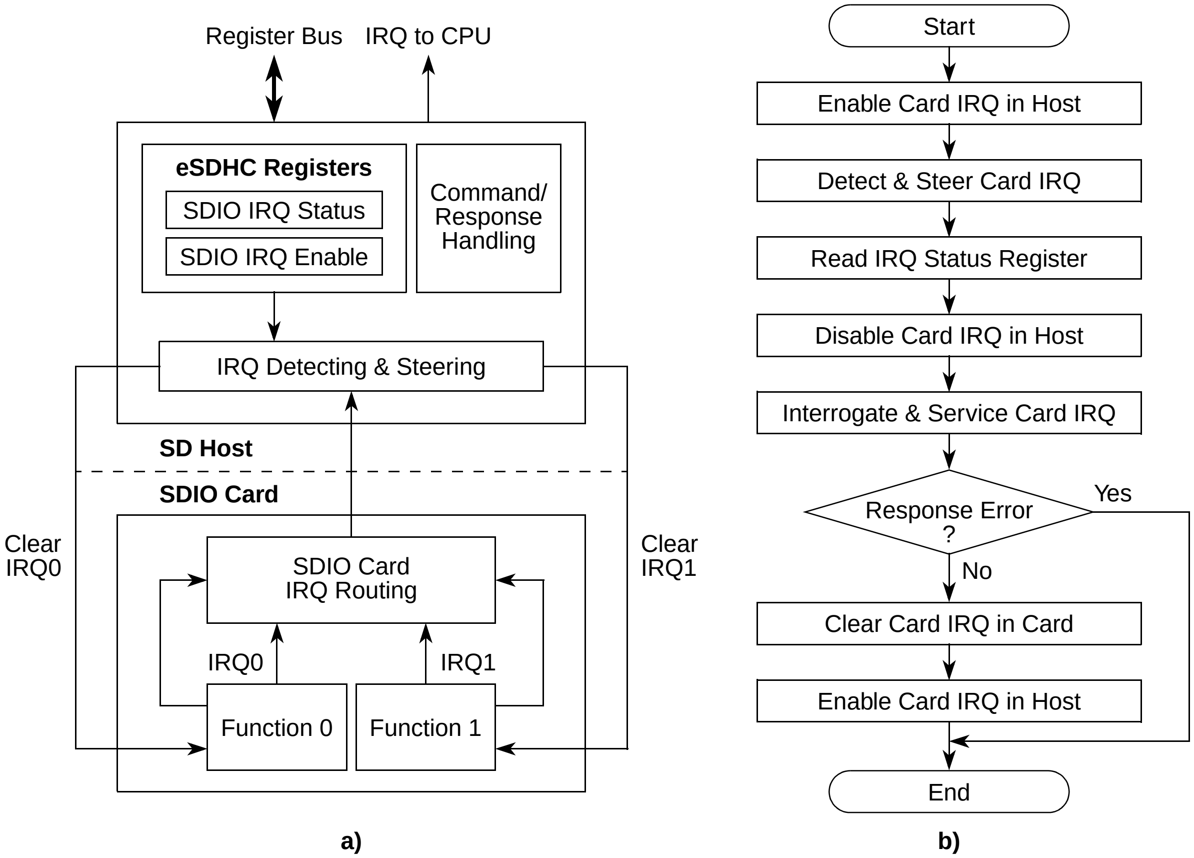


Figure 22-25. a) Card Interrupt Scheme; b) Card Interrupt Detection and Handling Procedure

22.4.7 Card Insertion and Removal Detection

The eSDHC uses the SDHC_DAT[3] pin to detect card insertion or removal. When SDHC_DAT[3] pin is used for card detection, the chip level integration needs to pull-down this pad as a default state. When there is no card on the MMC/SD bus, the SDHC_DAT[3] is pulled to a low voltage level by default. When any card is inserted to or removed from the socket, the eSDHC detects the logic value changes on the SDHC_DAT[3] pin and generates an interrupt.

22.4.8 Power Management and Wake-Up Events

When there is no operation between eSDHC and the card through SD bus, you can completely disable the internal clocks in the chip level clock control module to save power. When you need to use the eSDHC to communicate with the card, it can enable the clock and start the operation. This can be done by clearing the appropriate bit in the PPMR register as described in Chapter 8, "Power Management".

In some circumstances, when the clocks to eSDHC are disabled, or when system is in low power mode, there are some events when you need to enable the clock and handle the event. These events are called wakeup interrupts. The eSDHC can generate these interrupts even there are no clocks enabled. The three interrupts which can be used as wake-up events are:

- Card removal interrupt
- Card insertion interrupt
- SDIO card interrupt

These three wake-up events (or wake-up interrupts) can also wake up the system from low-power modes.

22.4.8.1 Setting Wake Up Events

For the eSDHC to respond to a wake up event, the software must set the respective wake up enable bit before the CPU enters sleep mode. Refer to Section 22.3.8, "Protocol Control Register (PROCTL)," for more information on the wakeup enable bits.

Before the software disables the host clock, it should ensure that all of the following conditions have been met:

- No read or write transfer is active
- Data and command lines are not active
- No interrupts are pending
- Internal data buffer is empty

22.5 Initialization/Application Information

All communication between system and cards are controlled by the host. The host sends commands of two types: broadcast and addressed (point-to-point) commands.

Broadcast commands are intended for all cards, such as GO_IDLE_STATE, SEND_OP_COND, ALL_SEND_CID, etc. In broadcast mode, all cards are in the open-drain mode to avoid bus contention.

Refer to Section 22.5.5, “Commands for MMC/SD/SDIO/CE-ATA,” for the commands of bc and bcr categories.

After the broadcast command CMD3 is issued, the cards enter standby mode. Addressed type commands are used from this point. In this mode, the SDHC_CMD/SDHC_DAT I/O pads turn to push-pull mode, to have the driving capability for maximum frequency operation. Refer to Section 22.5.5, “Commands for MMC/SD/SDIO/CE-ATA,” for the commands of ac and adtc categories.

22.5.1 Command Send and Response Receive Basic Operation

Assuming data type WORD is an unsigned 32-bit integer, the below flow is a guideline for sending a command to the card(s):

```
send_command(cmd_index, cmd_arg, other requirements)
{
WORD wCmd; // 32-bit integer to make up the data to write into the XFERTYP register, it is
          // recommended to implement in a bit-field manner
wCmd = (<cmd_index> & 0x3f) >> 24; // set the first 8 bits as ‘00’+<cmd_index>
set CMDTYP, DPSEL, CICEN, CCCEN, RSTTYP, and DTDSEL according to the command index;
         // XFERTYP register bits
if (internal DMA is used) wCmd |= 0x1;
if (multi-block transfer) {
         set XFERTYP[MSBSEL] bit;
         if (finite block number) {
                  set XFERTYP[BCEN] bit;
                  if (auto12 command is to use) set XFERTYP[AC12EN] bit;
         }
}
write_reg(CMDARG, <cmd_arg>); // configure the command argument
write_reg(XFERTYP, wCmd); // set XFERTYP register as wCmd value to issue the command
}
wait_for_response(cmd_index)
{
while (IRQSTAT[CC] is not set); // wait until command complete bit is set
read IRQSTAT register and check if any error bits about command are set;
if (any error bits are set) report error;
write 1 to clear IRQSTAT[CC] and all command error bits;
}
```

For the sake of simplicity, the function wait_for_response is implemented here by means of polling. For an effective and formal way, the response is usually checked after the command complete interrupt is received. By doing this, ensure the corresponding interrupt status bits are enabled.

For some scenarios, the response timeout is expected. For instance, after all cards respond to CMD3 and go to the standby state, no response to the host when CMD2 is sent. The host driver should manage false errors similar to this with caution.

22.5.2 Card Identification Mode

When a card is inserted to the socket or the card was reset by the host, the host needs to validate the operation voltage range, identify the cards, and request the cards to publish the relative card address (RCA) or to set the RCA for the MMCs. For CE-ATA cards, the device is connected by a point-to-point manner, and no RCA is needed. All data communications in the card identification mode use the command line (SDHC_CMD) only. Refer to CE-ATA digital protocol revision 1.1 for details.

22.5.2.1 Card Detect

See Figure 22-26 for a flow diagram showing the detection of MMC, SD, and SDIO cards using the eSDHC.

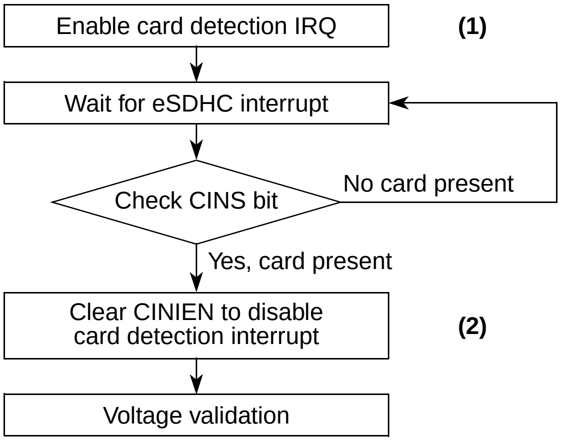


Figure 22-26. Flow Diagram for Card Detection

- Set IRQSIGEN[CINIEN] to enable card detection interrupt.
- When an interrupt from eSDHC is received, check IRQSTAT[CINS] to see if it is caused by card insertion.
- Clear the IRQSIGEN[CINIEN] to disable card detection interrupt and ignore all card insertion interrupt afterwards.

To detect a CE-ATA device:

1. Complete the normal MMC reset and initialization procedures
2. The host driver should issue CMD60 to check CE-ATA signature
3. If the device responds to the command with the CE-ATA signature, a CE-ATA device has been found.
4. The driver should query EXT_CSD register byte 504 (S_CMD_SET) in MMC register space.
5. If the ATA bit (bit 4) is set, then the MMC device is an ATA device.
   a) The driver should set the ATA bit (bit 4) of the EXT_CSD register byte 191 (CMD_SET) to activate the ATA command set for use. To choose the command set, the driver should issue CMD6.
6. If the ATA bit (bit 4) is cleared, the device does not support ATA mode, and the driver should not issue ATA command to the device.

22.5.2.2 Reset

The host consists of three types of reset:

- Hardware reset (card and host) which is driven by POR (power on reset).
- Software reset (host only) is proceeded by the write operation on the SYSCTL[RSTD], SYSCTL[RSTC], or SYSCTL[RSTA] bits to reset the data part, command part, or all parts of the host controller, respectively.
- Card reset (card only). The command CMD0, GO_IDLE_STATE, is the software reset command for all types of CE-ATA cards, MMCs, and SD memory cards. This command sets each card into idle state regardless of the current card state. For an SDIO card, CMD52 is used to write I/O reset in CCCR. The cards are initialized with a default relative card address (RCA = 0x0000) and with a default driver stage register setting (lowest speed, highest driving current capability).

After the card is reset, the host needs to validate the voltage range of the card. See Figure 22-27 for the software flow to reset the eSDHC and card.

For a CE-ATA device that supports ATA mode, before issuing CMD0 to reset MMC layer, two CMD39 should be issued back-to-back to the ATA control register.

- First CMD39 should have the SRST bit of the CE-ATA register set (defined in the CE-ATA Specification).
- Second CMD39 command should have the SRST bit cleared .

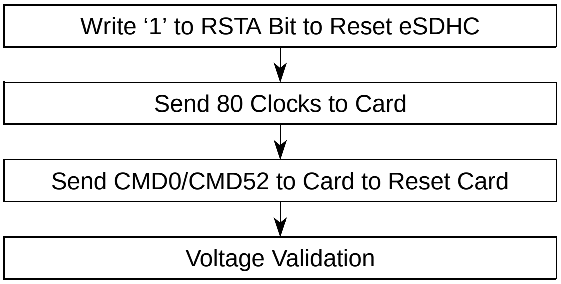

Write ‘1’ to RSTA Bit to Reset eSDHC
Send 80 Clocks to Card
Send CMD0/CMD52 to Card to Reset Card
Voltage Validation

Figure 22-27. Flow Chart for Reset of eSDHC and SD I/O Card

```
software_reset()
{
         set_bit(SYSCTL, RSTA);              // software reset the host
         set SYSCTL[DTOCV and SDCLKFS];      // get the SDHC_CLK of frequency around 400 KHz
         configure I/O pad;                  // set the voltage of external card to around 3.0 V
         poll PRSSTAT[CIHB and CDIHB];       // wait until both bits are cleared
         set_bit(SYSCTRL, INTIA);            // send 80 clock ticks for card to power-up
         send_command(CMD_GO_IDLE_STATE, <other parameters>); // reset the card with CMD0
         or send_command(CMD_IO_RW_DIRECT, <other parameters>);
}
```

22.5.2.3 Voltage Validation

All cards should be able to establish communication with the host using any operation voltage in the maximum allowed voltage range specified in this standard. However, the supported minimum and maximum values for V_{DD} are defined in the operation conditions register (OCR) and may not cover the whole range. Cards that store the CID (card identification) and CSD data in the preloaded memory are only able to communicate these information under data transfer V_{DD} conditions. This means that if the host and card have different V_{DD} ranges, the card is not able to complete the identification cycle, nor is it able to send CSD data.

Therefore, a special command is available:

- SEND_OP_CONT (CMD1 for MMC and CE-ATA),
- SD_SEND_OP_CONT (ACMD41 for SD Memory), and
- IO_SEND_OP_CONT (CMD5 for SD I/O).

The voltage validation procedure is designed to provide a mechanism to identify and reject cards which do not match the V_{DD} range(s) desired by the host. This is accomplished by the host sending the desired V_{DD} voltage window as the operand of this command. Cards that can not perform data transfer in the specified range must discontinue any further bus operations and enter the inactive state. By omitting the voltage range in the command, the host can query each card and determine the common voltage range before sending out-of-range cards into the inactive state. This query should be used if the host is able to select a common voltage range or if a notification should be sent to the system when a non-usable cards in the stack is detected.

The following steps illustrate how to perform voltage validation when a card is inserted:

```
voltage_validation(voltage_range_arguement)
{
label the card as UNKNOWN;
send_command(IO_SEND_OP_COND, 0x0, <other parameters are omitted>);
         // CMD5, check SDIO operation voltage, command argument is zero
if (RESP_TIMEOUT != wait_for_response(IO_SEND_OP_COND)) { // SDIO command is accepted
         if (0 < number of IO functions) {
                  label the card as SDIO;
                  IORDY = 0;
                  while (!(IORDY in IO OCR response)) { // set voltage range for each IO function
                           send_command(IO_SEND_OP_COND, <voltage range>, <other parameter>);
                           wait_for_response(IO_SEND_OP_COND);
                  } // end of while ...
         } // end of if (0 < ...)
         if (memory part is present inside SDIO card) label the card as SDCombo;
                  // this is an SD-Combo card
} // end of if (RESP_TIMEOUT...)
if (the card is labeled as SDIO card) return;
         // card type is identified and voltage range is set, so exit the function;
send_command(APP_CMD, 0x0, <other parameters are omitted>);
         // CMD55, application specific CMD prefix
if (no error calling wait_for_response(APP_CMD, <...>) { // CMD55 is accepted
         send_command(SD_APP_OP_COND, <voltage range>, <...>);
                  // ACMD41, to set voltage range for memory part or SD card
         wait_for_response(SD_APP_OP_COND); // voltage range is set
         if (card type is UNKNOWN) label the card as SD;
         return;
} // end of if (no error ...
else if (errors other than timeout occur) { // command/response pair is corrupted
         respond to it by program specific manner;
} // of else if (response timeout)
else { // CMD55 is refused, it must be MMC or CE-ATA card
         if (card is already labeled as SD Combo) { // change label
                  re-label the card as SDIO;
                  ignore the error or report it;
                  return; // card is identified as SDIO card
         } // of if (card is ...)
         send_command(SEND_OP_COND, <voltage range>, <...>);
         if (RESP_TIMEOUT == wait_for_response(SEND_OP_COND)) {
                                   // CMD1 is not accepted, either
                  label the card as UNKNOWN;
                  return;
         } // of if (RESP_TIMEOUT...)
         if (check for CE-ATA signature succeeded) { // the card is CE-ATA
                  store CE-ATA specific info from the signature;
                  label the card as CE-ATA;
         } // of if (check for CE-ATA...)
         else label the card as MMC;
} // of else
}
```

22.5.2.4 Card Registry

Card registry on MMC and SD/SDIO/SD Combo cards are different. For CE-ATA, the card enters the transfer state after reset is complete.

For the SD card, the identification process starts at a clock rate lower than 400 KHz and the power voltage higher than 2.7 V, as defined by the card specification. At this time, the SDHC_CMD line output drives are push-pull drivers instead of open-drain. After the bus is activated, the host requests the card to send their valid operation conditions. The response to ACMD41 is the operation condition register of the card. The same command should be sent to all of the new cards in the system. Incompatible cards are placed into the inactive state. The host then issues the command, ALL_SEND_CID (CMD2), to each card to get its CID. Cards that are currently unidentified (that is, in ready state), send their CID number as the response. After the CID is sent by the card, the card goes into the identification state.

The host then issues Send_Relative_Addr (CMD3), requesting the card to publish a new relative card address (RCA) that is shorter than CID. This RCA is used to address the card for future data transfer operations. Once the RCA is received, the card changes its state to the standby state. At this point, if the host wants the card to have an alternative RCA number, it may ask the card to publish a new number by sending another Send_Relative_Addr command to the card. The last published RCA is the actual RCA of the card.

The host repeats the identification process with CMD2 and CMD3 for each card in the system until the last CMD2 gets no response from any of the cards in system.

For MMC and CE-ATA operation, the host starts the card identification process in open-drain mode with the identification clock rate lower than 400 KHz, the power voltage higher than 2.7 V. The open-drain driver stages on the SDHC_CMD line allow parallel card operation during card identification. After the bus is activated the host requests the cards to send their valid operation conditions (CMD1). The response to CMD1 is the wired-OR operation on the condition restrictions of all cards in the system. Incompatible cards are sent into inactive state. The host then issues the broadcast command All_Send_CID (CMD2), asking all cards for their unique CID number. All unidentified cards (the cards in ready state) simultaneously start sending their CID numbers serially, while bit-wise monitoring their outgoing bit stream. Those cards, whose outgoing CID bits do not match the corresponding bits on the command line in any one of the bit periods, stop sending their CID immediately and must wait for the next identification cycle. Since the CID is unique for each card, only one card can successfully send its full CID to the host.

This card then goes into identification state. Thereafter, the host issues Set_Relative_Addr (CMD3) to assign to this card a relative card address (RCA). Once the RCA is received, the card state changes to the stand-by state, and the card does not react in further identification cycles, and its output driver switches from open-drain to push-pull. The host repeats the process, namely CMD2 and CMD3, until the host receives a time-out condition to recognize completion of the identification process.

```
card_registry()
{
do { // decide RCA for each card until response timeout
         if(card is labeled as SD Combo or SDIO) { // for SDIO card like device
                  send_command(SET_RELATIVE_ADDR, 0x00, <...>);
                           // ask SDIO card to publish its RCA
                  retrieve RCA from response;
         } // end if (card is labeled as SD Combo...)
         else if (card is labeled as SD) { // for SD card
                  send_command(ALL_SEND_CID, <...>);
                  if (RESP_TIMEOUT == wait_for_response(ALL_SEND_CID)) break;
                  send_command(SET_RELATIVE_ADDR, <...>);
                  retrieve RCA from response;
         } // else if (card is labeled as SD ...)
         else if (card is labeled as MMC or CE-ATA) { // treat CE-ATA as MMC
                  send_command(ALL_SEND_CID, <...>);
                  rca = 0x1; // arbitrarily set RCA, 1 here for example, this RCA is also the
                            // relative address to access the CE-ATA card
                  send_command(SET_RELATIVE_ADDR, 0x1 << 16, <...>);
                           // send RCA at upper 16 bits
         } // end of else if (card is labeled as MMC...)
} while (response is not timeout);
}
```

22.5.3 Card Access

These sections describe the supported access modes with external cards.

22.5.3.1 Block Write

This section describes the process of writing data to external cards in block mode.

22.5.3.1.1 Normal Write

During block write (CMD24–27, CMD60, CMD61), one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. If the CRC fails, the card should indicate the failure on the SDHC_DAT line (see below). The transferred data is discarded and not written, and all further transmitted blocks (in multi-block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed (CSD parameter WRITE_BLK_MISALIGN is not set and the CE-ATA card does not support partial block write), the card detects the block misalignment error and aborts programming before the beginning of the first misaligned block. The card sets the ADDRESS_ERROR error bit in the status register, defined in the MMC/SD Specification, and then waits in the receive-data state for a stop command while ignoring all further data transfers. For CE-ATA cards, consult the CE-ATA card specification for behavior during a block misalignment. The write operation is also aborted if the host attempts to write over a write-protected area.

For MMC and SD cards, programming the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in the ROM, this unchangeable section must match the corresponding section of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require a long and unpredictable period of time to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing. If its write buffer is full and unable to accept new data from a new WRITE_BLOCK command, the card holds the SDHC_DAT line low. The host may poll the status of the card with a SEND_STATUS command (CMD13) or other means for SDIO and CE-ATA cards, at any time and the card responds with its status. The card status indicates whether the card can accept new data or if the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card) to change the card into the standby state and release the SDHC_DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling SDHC_DAT low if programming is still in progress and the write buffer is unavailable.

For simplicity, the software flow described below incorporates the internal DMA, and the write operation is a multi-block write with Auto CMD12 enabled. For the other two methods (external DMA or CPU polling status) and different transfer nature, the internal DMA part of the procedure should be removed and alternative steps inserted.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
   - — MMC/SD cards — use SET_BLOCKLEN (CMD16)
   - — SDIO cards or the I/O portion of SD Combo cards — IO_RW_DIRECT (CMD52) to set I/O block size bit field in the CCCR register (for function 0) or FBR (for functions 1–7)
   - — CE-ATA cards — Configure bits 1–0 of scrControl register, defined in the CE-ATA specification, to set the block size
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

22.5.3.1.2 Write with Pause

The write operation can be paused during the transfer. Instead of stopping the SDHC_CLK at any time to pause all the operations which is also inaccessible to the host driver, the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Since there is no timeout condition in a write operation during the data blocks, a write operation to the cards can be paused in this way and if line SDHC_DAT0 is not required to de-assert to release busy state, no suspend command is needed.

Similar to the flow described in Section 22.5.3.1.1, “Normal Write,” the write with pause is shown with the same type of write operations:

1. Check the card status and wait until card is ready for data.
2. Set the card block length.
   — MMC/SD cards — use SET_BLOCKLEN (CMD16)
   — SDIO cards or the I/O portion of SD Combo cards — use IO_RW_DIRECT (CMD52) to set the I/O block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
   — CE-ATA cards — configure bits 1–0 of scrControl register to set the block size
3. Set the eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Set PROCTL[SABGREQ].
7. Wait for the transfer complete interrupt.
8. Clear PROCTL[SABGREQ].
9. Check the status bit to see if a read CRC error occurred.
10. Set PROCTL[CREQ] to continue the read operation.
11. Wait for the transfer complete interrupt.
12. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

The number of blocks left during the data transfer is accessible by reading the content of BLKATTR[BLKCNT]. Due to the data transfers and setting PROCTL[SABGREQ] are concurrent, along with the delay of register read and the register setting, the actual number of blocks left may not be the same as the value read earlier. The driver should read the value of BLKATTR[BLKCNT] after the transfer is paused and the transfer complete interrupt is received.

It is also possible that the transfer of the last block begins when the stop-at-block-gap request is sent to the buffer. In this case, the next block gap is the actual end of the transfer, and therefore, the request is ignored. The driver should treat this as a non-pause transfer and a common write operation.

When the write operation is paused, the data transfer inside the host system does not stop and the transfer remains active until the data buffer is full. Therefore, avoid using the suspend command for the SDIO card. When such command is sent, the eSDHC assumes the system switches to another function of the SDIO card and flushes the data buffer. The eSDHC reads the resume command as a normal command with a data transfer, and it is the driver’s responsibility to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN, AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of multi-block transfer.

22.5.3.2 Block Read

22.5.3.2.1 Normal Read

For block reads, the basic unit of a data transfer is a block whose maximum size is stored in areas defined in corresponding card specifications. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17, CMD18, CMD53, CMD60, CMD61, and so on, can initiate a block read. After completing the transfer, the card returns to the transfer state.

For multi-block reads, data blocks are continuously transferred until a stop command is issued. If the host uses partial blocks whose accumulated length is not block aligned and block misalignment is not allowed, the card which does not support partial block length, should detect the block misalignment at the beginning of the first misaligned block and report the error, depending on its card type.

For simplicity, the software flow described below incorporates the internal DMA, and the read operation is a multi-block read with Auto CMD12 enabled. For the other two methods (external DMA or CPU polling status) and different transfer nature, the internal DMA part should be removed and the alternative steps are straightforward.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
   - — MMC/SD cards — use SET_BLOCKLEN (CMD16)
   - — SDIO cards or the I/O portion of SD Combo cards — use IO_RW_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
   - — CE-ATA cards — configure bits 1–0 of scrControl register to set the block size
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

22.5.3.2.2 Read with Pause

In general, the read operation is not able to pause. Only the SDIO card (and SD Combo card working under I/O mode) supporting the read wait feature can pause during the read operation. If the SDIO card supports read wait (CCCR[SRW] = 1), the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Before setting SABGREQ, PROCTL[RWCTL] must be set. Otherwise, the eSDHC does not assert the read wait signal during the block gap and data corruption occurs. It is recommended to set the RWCTL bit once the read wait capability of the SDIO card is recognized.

Similar to the flow described in Section 22.5.3.2.1, “Normal Read,” the read with pause is shown with the same type of read operations:

1. Check CCCR[SRW] in the SDIO card to confirm the card supports read wait.
2. Set PROCTL[RWCTL].

3. Check the card status and wait until the card is ready for data.
4. Set the card block length.
   - — MMC/SD cards — use SET_BLOCKLEN (CMD16)
   - — SDIO cards or the I/O portion of SD Combo cards — use IO_RW_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
   - — CE-ATA cards — configure bits 1–0 of scrControl register to set the block size.
5. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in Step 2.
6. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
7. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
8. Set PROCTL[SABGREQ].
9. Wait for the transfer complete interrupt.
10. Clear PROCTL[SABGREQ].
11. Check the status bit to see if a read CRC error occurred.
12. Set PROCTL[CREQ] to continue the read operation.
13. Wait for the transfer complete interrupt.
14. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

Similar to the write operation, it is possible to meet the ending block of the transfer when paused. In this case, the eSDHC ignores the stop-at-block-gap request and treats it as a command read operation.

Unlike the write operation, there is no remaining data inside the buffer when the transfer is paused. All data received before the pause is transferred to the host system. Whether or not a suspend command is sent, the internal data buffer is not flushed.

If the suspend command is sent and the transfer is later resumed by means of the resume command, the eSDHC takes the command as a normal one accompanied with data transfer, and it is left for the driver to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN] and IRQSTT[AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of a multi-block transfer.

22.5.3.3 Transfer Error

22.5.3.3.1 CRC Error

At the end of a block transfer, a write CRC status error or read CRC error may occur. For this type of error, the last block received should be discarded because the integrity of the data block is not guaranteed. It is recommended to discard the following data blocks and re-transfer the block from the corrupted one. For a multi-block transfer, the host driver should issue CMD12 to abort the current process and start the transfer by a new data command. In this scenario, even when the XFERTYP[AC12EN, BCEN] are set, the eSDHC does not automatically send CMD12 because the last block is not transferred. On the other hand, if it is within the last block that CRC error occurs, Auto CMD12 is sent by the eSDHC. In this case, the driver should resend or re-obtain the last block with a single block transfer.

22.5.3.3.2 Internal DMA Error

During the data transfer with the internal DMA, if the DMA engine encounters an error on the CSB bus, the DMA operation is aborted and a DMA error interrupt is sent to the host system. When acknowledged by such an interrupt, the driver should calculate the start address of the data block where the error occurred. The start address can be calculated by either of the following methods:

- Read the DSADDR[DSADDR] field. The error occurs during the previous burst. Therefore, by taking the block size, the previous burst length, and the start address of the next burst transfer into account, one can obtain the start address of the corrupted block.
- Read the BLKATTR[BLKCNT] field. The start address of the corrupted block can be calculated by the number of blocks left, the total number to transfer, the start address of transfer, and the size of each block. However, if BCEN is not set, the contents of the block attribute register does not change and this method does not work.

When a DMA error occurs, it is recommended to abort the current transfer by means of CMD12 (for multi-block transfer), apply a reset for data, and restart the transfer from the corrupted block to recover the error.

22.5.3.3.3 Auto CMD12 Error

After the last block of a multi-block transfer is sent or received and XFERTYP[AC12EN] is set when the data transfer is initiated by the data command, the eSDHC automatically sends CMD12 to the card to stop the transfer. When an error occurs at this point, it is recommended that the host driver responds by:

1. Auto CMD12 response timeout. It is not certain whether the command has been accepted by the card or not. The driver should clear the Auto CMD12 error status bits and resend CMD12 until it is accepted by the card.
2. Auto CMD12 response CRC error. Since CMD12 has been received by the card, the card aborts the transfer. The driver may ignore the error and clear the error status bit.
3. Auto CMD12 conflict error or not sent. The command was not sent. Therefore, the driver should send CMD12 manually.

22.5.3.4 Card Interrupt

The external cards can inform the host controller through the use of special signals. For SDIO cards, it can be the low level on the SDHC_DAT[1] line during a specific period. For CE-ATA cards, it can be a pulse on the SDHC_CMD line to inform the host controller that a command and its response have been completed. It is possible some other external interrupt behaviors can be defined. The eSDHC only monitors the SDHC_DAT[1] line and supports SDIO interrupts.

When an SDIO interrupt is captured by the eSDHC and the host system is informed by the eSDHC asserting its interrupt line, the interrupt service of host driver is requested.

As the interrupt source is controlled by the external card, the interrupt from the SDIO card must be served before the CINT bit is cleared. Refer to Section 22.4.6.3, “Card Interrupt Handling,” for the card interrupt handling flow.

22.5.4 Switch Function

MMCs transferring data with a bus width other than one-bit wide is a new feature added to the MMC specification. The high-speed timing mode for all card devices is also newly-defined in recent various card specifications. To enable these new features, a type of switch command should be issued by the host driver.

NOTE

High-speed mode is not supported on this device.

For SDIO cards, the high speed mode is enabled by writing to CCCR[EHS] after the CCCR[SHS] bit is confirmed. For SD cards, the high-speed mode is queried and enabled by CMD6 (with the mnemonic symbol as SWICH_FUNC); for MMCs (and CE-ATA cards over the MMC interface), the high-speed mode is queried by CMD8 and enabled by CMD6 (with the mnemonic symbol as SWITCH).

The 4-bit and 8-bit bus width of MMC is also enabled by the SWITCH command, but with a different argument.

These new functions can also be disabled by software reset (for SDIO card, by setting RES bit in CCCR register; for other cards, by issuing CMD0), but such manner of restoring to normal mode is not recommended because a complete identification process is needed before the card is ready for data transfer.

For simplicity, the following flowcharts do not show a current capability check, which is recommended in the function switch process.

22.5.4.1 Query, Enable and Disable SDIO High Speed Mode

```
enable_sdio_high_speed_mode(void)
{
send CMD52 to query bit SHS at address 0x13;
if (SHS bit is ‘0’)
        {
        report the SDIO card does not support high speed mode and return;
        }
send CMD52 to set bit EHS at address 0x13 and read after write to confirm EHS bit is set;
change clock divisor value or configure the system clock feeding into eSDHC to generate the
card_clk of around 50MHz;
(data transactions like normal peers)
}
disable_sdio_high_speed_mode(void)
{
send CMD52 to clear bit EHS at address 0x13 and read after write to confirm EHS bit is cleared;
change clock divisor value or configure the system clock feeding into eSDHC to generate the
card_clk of the desired value below 25MHz;
(data transactions like normal peers)
}
```

22.5.4.2 Query, Enable and Disable SD High Speed Mode

```
enable_sd_high_speed_mode(void)
{
        set BLKATTR[BLKCNT] to 1 (block), set BLKATTR[BLKSIZE] to 64 (bytes);
        send CMD6, with argument 0xFFFFF1 and read 64 bytes of data accompanying the R1
                response;
        wait data transfer done bit is set;
        check if the bit 401 of received 512 bit is set;
        if (bit 401 is ‘0’) report the SD card does not support high speed mode and return;
        send CMD6, with argument 0x80FFFFF1 and read 64 bytes of data accompanying the R1
                 response;
        check if the bit field 379~376 is 0xF;
        if (the bit field is 0xF) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                 the card_clk of around 50MHz;
        (data transactions like normal peers)
}
disable_sd_high_speed_mode(void)
{
        set BLKCNT field to 1 (block), set BLKSIZE field to 64 (bytes);
        send CMD6, with argument 0x80FFFFF0 and read 64 bytes of data accompanying the R1
                 response;
        check if the bit field 379~376 is 0xF;
        if (the bit field is 0xF) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                 the card_clk of the desired value below 25MHz;
        (data transactions like normal peers)
}
```

22.5.4.3 Query, Enable and Disable MMC High Speed Mode

```
enable_mmc_high_speed_mode(void)
{
        send CMD9 to get CSD value of MMC;
        check if the value of SPEC_VER field is 4 or above;
        if (SPEC_VER value is less than 4) report the MMC does not support high speed mode and
                 return;
        set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
        send CMD8 to get EXT_CSD value of MMC;
        extract the value of CARD_TYPE field to check the ‘high speed mode’ in this MMC is
                 26MHz or 52MHz;
        send CMD6 with argument 0x1B90100;
        send CMD13 to wait card ready (busy line released);
        send CMD8 to get EXT_CSD value of MMC;
        check if HS_TIMING byte (byte number 185) is 1;
        if (HS_TIMING is not 1) report MMC switching to high speed mode failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                 the card_clk of around 26MHz or 52MHz according to the CARD_TYPE;
        (data transactions like normal peers)
}

disable_mmc_high_speed_mode(void)
{
        send CMD6 with argument 0x2B90100;
        set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
        send CMD8 to get EXT_CSD value of MMC;
        check if HS_TIMING byte (byte number 185) is 0;
        if (HS_TIMING is not 0) report the function switch failed and return;
        change clock divisor value or configure the system clock feeding into eSDHC to generate
                 the card_clk of the desired value below 20MHz;
        (data transactions like normal peers)
}
```

22.5.4.4 Set MMC Bus Width

```
change_mmc_bus_width(void)
{
         send CMD9 to get CSD value of MMC;
         check if the value of SPEC_VER field is 4 or above;
         if (SPEC_VER value is less than 4) report the MMC does not support multiple bit width
                  and return;
         send CMD6 with argument 0x3B70x00; (8-bit, x=2; 4-bit, x=1; 1-bit, x=0)
         send CMD13 to wait card ready (busy line released);
         (data transactions like normal peers)
}
```

22.5.5 Commands for MMC/SD/SDIO/CE-ATA

See Table 22-26 for the list of commands for the MMC/SD/SDIO cards. Refer to the corresponding specifications for details about the command information.

Four kinds of commands control the MMC:

1. Broadcast commands (bc)—no response
2. Broadcast commands with response (bcr)—response from all cards simultaneously
3. Addressed (point-to-point) commands (ac)—no data transfer on SDHC_DAT
4. Addressed (point-to-point) data transfer commands (ADTC)

Table 22-26. Commands for MMC/SD/SDIO/CE-ATA

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD0	bc	[31:0] stuff bits	—	GO_IDLE_STATE	Resets all MMC and SD memory cards to idle state.
CMD1	bcr	[31:0] OCR without busy	R3	SEND_OP_COND	Asks all MMCs and SD memory cards in idle state to send their operation conditions register contents in the response on the SDHC_CMD line.
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the SDHC_CMD line.
CMD3[(1)]	ac	[31:6] RCA [15:0] stuff bits	R1 R6(SDIO)	SET/SEND_RELATIVE_ADDR	Assigns relative address to the card.
CMD4	bc	[31:0] DSR [15:0] stuff bits	—	SET_DSR	Programs the DSR of all cards.
CMD5	bc	[31:0] OCR without busy	R4	IO_SEND_OP_COND	Asks all SDIO cards in idle state to send their operation conditions register contents in the response on the SDHC_CMD line.

Table 22-26. Commands for MMC/SD/SDIO/CE-ATA (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD6[(2)]	adtc	[31] Mode 0: Check function 1: Switch function [30:8] Reserved for function groups 6 ~ 3 (All 0 or 0xFFFF) [7:4] Function group1 for command system [3:0] Function group2 for access mode	R1	SWITCH_FUNC	Checks switch ability (mode 0) and switch card function (mode 1). Refer to SD Physical Specification version 1.1 for details.
CMD6[(3)]	ac	[31:26] Set to 0 [25:24] Access [23:16] Index [15:8] Value [7:3] Set to 0 [2:0] Cmd Set	R1b	SWITCH	Switches the mode of operation of the selected card or modifies the EXT_CSD registers. Refer to the MultiMediaCard System Specification version 4.0 final draft 2 for details.
CMD7	ac	[31:6] RCA [15:0] stuff bits	R1b	SELECT/DESELECT_CARD	Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all.
CMD8	adtc	[31:0] stuff bits	R1	SEND_EXT_CSD	The card sends its EXT_CSD register as a block of data, with block size of 512 bytes.
CMD9	ac	[31:6] RCA [15:0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the SDHC_CMD line.
CMD10	ac	[31:6] RCA [15:0] stuff bits	R2	SEND_CID	Addressed card sends its card-identification (CID) on the SDHC_CMD line.
CMD11	adtc	[31:0] data address	R1	READ_DAT_UNTIL_STOP	Reads data stream from the card starting at the given address until STOP_TRANSMISSION is received.
CMD12	ac	[31:0] stuff bits	R1b	STOP_TRANSMISSION	Forces the card to stop transmission.
CMD13	ac	[31:6] RCA [15:0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14	Reserved				
CMD15	ac	[31:6] RCA [15:0] stuff bits	—	GO_INACTIVE_STATE	Sets the card to inactive state in order to protect the card stack against communication breakdowns.
CMD16	ac	[31:0] block length	R1	SET_BLOCKLEN	Sets the block length (in bytes) for all following block commands (read and write). Default block length is specified in the CSD.

Table 22-26. Commands for MMC/SD/SDIO/CE-ATA (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD17	adtc	[31:0] data address	R1	READ_SINGLE_BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	adtc	[31:0] data address	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a stop command.
CMD19	Reserved				
CMD20	adtc	[31:0] data address	R1	WRITE_DAT_UNTIL_STOP	Writes data stream from the host starting at the given address until the STOP_TRANSMISION command is received.
CMD21–23	Reserved				
CMD24	adtc	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	adtc	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until the STOP_TRANSMISSION command is received.
CMD26	adtc	[31:0] stuff bits	R1	PROGRAM_CID	Programming of the card identification register. This command should be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for the manufacturer.
CMD27	adtc	[31:0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28	ac	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write-protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE).
CMD29	ac	[31:0] data address	R1b	CLR_WRITE_PROT	If the card provides write-protection features, this command clears the write protection bit of the addressed group.
CMD30	adtc	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card provides write-protection features, this command asks the card to send the status of the write-protection bits.
CMD31	Reserved				
CMD32	ac	[31:0] data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group.
CMD33	ac	[31:0] data address	R1	TAG_SECTOR_END	Sets the address of the last write block of the continuous range to be erased.

Table 22-26. Commands for MMC/SD/SDIO/CE-ATA (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD34	ac	[31:0] data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection.
CMD35	ac	[31:0] data address	R1	TAG_ERASE_GROUP_STA RT	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31:0] data address	R1	TAG_ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	ac	[31:0] data address	R1	UNTAG_ERASE_GROUP	Removes one previously selected erase group from the erase selection.
CMD38	ac	[31:0] stuff bits	R1b	ERASE	Erase all previously selected sectors.
CMD39	ac	[31:0] RCA [15] register write flag [14:8] register address [7:0] register data	R4	FAST_IO	Used to write and read 8-bit (register) data fields. The command address a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the address register. This command accesses application dependent registers which are not defined in the MMC standard.
CMD40	bcr	[31:0] stuff bits	R5	GO_IRQ_STATE	Sets the system into interrupt mode.
CMD41	Reserved				
CDM42	adtc	[31:0] stuff bits	R1b	LOCK_UNLOCK	Used to set/reset the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
CMD43–51	Reserved				
CMD52	ac	[31:0] stuff bits	R5	IO_RW_DIRECT	Access a single register within the total 128 Kbytes of register space in any I/O function.
CMD53	ac	[31:0] stuff bits	R5	IO_RW_EXTENDED	Access a multiple I/O register with a single command, it allows the reading or writing of a large number of I/O registers.
CMD54	Reserved				
CMD55	ac	[31:16] RCA [15:0] stuff bits	R1	APP_CMD	Indicates to the card that the next command is an application specific command rather that a standard command.

Table 22-26. Commands for MMC/SD/SDIO/CE-ATA (continued)

CMD INDEX	Type	Argument	Resp	Abbreviation	Description[1]
CMD56	adtc	[31:1] stuff bits [0]: RD/WR	R1b	GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general-purpose or application-specific commands. The size of the data block is set by the SET_BLOCK_LEN command.
CMD57–59	Reserved				
CMD60	adtc	[31] WR [30:24] stuff bits [23:16] address [15:8] stuff bits [7:0] byte count	R1b	RW_MULTIPLE_REGISTER	CE-ATA devices contain a set of status and control registers that begin at register offset 0x80. These registers are used to control the behavior of the device and to retrieve status information regarding the operation of the device. All status and control registers are 32 bits in size (one word) and are word-aligned. CMD60 should be used to read and write to these registers.
CMD61	adtc	[31] WR [30:16] stuff bits [15:0] data unit count	R1b	RW_MULTIPLE_BLOCK	The host issues RW_MULTIPLE_BLOCK (CMD61) to begin the data transfer for the ATA command.
CMD62–63	Reserved				
ACMDs should be preceded with the APP_CMD command (Commands listed below are for SD cards only. Other SD commands not listed below are not supported by this module)					
ACMD6	ac	[31:2] stuff bits [1:0] bus width	R1	SET_BUS_WIDTH	Defines the data bus width (00 = 1 bit or 10 = 4 bit bus) to be used for data transfer. The allowed data bus widths are given in SCR register.
ACMD13	adtc	[31:0] stuff bits	R1	SD_STATUS	Send the SD memory card status.
ACMD22	adtc	[31:0] stuff bits	R1	SEND_NUM_WR_ SECTORS	Send the number of the written (without errors) sectors. Responds with 32 bit + CRC data block.
ACMD23	ac	—	R1	SET_WR_BLK_ERASE_ COUNT	—
ACMD41	bcr	[31:0] OCR	R3	SD_APP_OP_COND	Asks the accessed card to send its operating condition register (OCR) content in the response on the SDHC_CMD line.
ACMD42	ac		R1	SET_CLR_CARD_DETECT	—
ACMD51	adtc	[31:0] stuff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR)

[1] Registers mentioned in this table are SD card registers.

NOTE

- CMD3 differs for MMC and SD cards
  For MMC cards, CMD3 is referred to as SET_RELATIVE_ADDR and has a response type R1
  For SD cards, CMD3 is referred to as SEND_RELATIVE_ADDR and has a response type R6, with RCA inside
- Command SWITCH is for high-speed MMC cards as well CE-ATA cards over MMC interface. The index field can contain any value from 0–255, but only values 0–191 are valid. If the index value is in the 192–255 range, the card does not perform any modification and the status bit EXT_CSD[SWITCH_ERROR] is set. The access bits are shown in Table 22-27:

Table 22-27. EXT_CSD Access Modes

Bits	Access Name	Operation
00	Command set	The command set is changed according to the command set field of the argument
01	Set bits	The bits in the pointed byte are set, according to the set bits in the value field.
10	Clear bits	The bits in the pointed byte are cleared, according to the set bits in the value field.
11	Write byte	The value field is written into the pointed byte.

22.5.6 Software Restrictions

22.5.6.1 Initialization Active

The driver cannot set SYSCTL[INITA] when any of the command or data lines are active, so the driver must ensure both CDIHB and CIHB bits are cleared. To auto clear the INITA bit, the SDCLKEN bit must be set; otherwise, no clocks can go to the card and INITA never clears.

22.5.6.2 Software Polling Procedure

When polling read or write, once the software begins a buffer read or write, it must access exactly the number of times as set in the watermark level register. Moreover, if the block size is not the value in watermark level register (read and write respectively), the software must access exactly the remaining number of words at the end of each block.

For example, for a read operation, if the RD_WML is 4, indicating the watermark level is 16 bytes, block size is 40 bytes, and the block number is 2, then the access times for the burst sequence in the whole transfer process must be 4, 4, 2, 4, 4, 2.

22.5.6.3 Suspend Operation

To suspend the data transfer, the software must inform the eSDHC that the suspend command is successfully accepted. To achieve this, after the suspend command is accepted by the SDIO card, software must send another normal command marked as suspend command (XFERTYP[CMDTYP] = 01) to inform the eSDHC that the transfer is suspended.

If software must resume the suspended trasnfer, it should read the value in BLKCNT to save the remaining number of blocks before sending the normal command marked as suspend. Otherwise, on sending the suspend command, the eSDHC regards the current transfer as aborted and changes BLKCNT to its original value, instead of keeping the remained number of blocks.

22.5.6.4 DMA Address Setting

To configure DMA address register, when TC bit is set, the register always updates itself with the internal address value to support dynamic address synchronization. Software must ensure TC bit is cleared prior to configuring the DMA address register.

22.5.6.5 Data Port Access

The data port does not support parallel access. For example, during an external DMA access, it is not allowed to write any data to the data port by CPU. During a CPU read operation, it is also prohibited to write any data to the data port, by either CPU or external DMA. Otherwise, the data is corrupted inside the eSDHC buffer.

22.5.6.6 Change Clock Frequency

The eSDHC does not automatically gate off the card clock when the host driver changes the clock frequency. To remove a possible glitch on the card clock, clear SYSCTL[SDCLKEN] when changing the clock divisor value, and set SDCLKEN after PRSSTAT[SDSTB] sets.

Chapter 23
Cryptographic Acceleration Unit (CAU)

23.1 Introduction

The cryptographic acceleration unit (CAU) is a ColdFire coprocessor implementing a set of specialized operations in hardware to increase the throughput of software-based encryption and hashing functions.

23.1.1 Block Diagram

Figure 23-1 shows a simplified block diagram of the CAU.

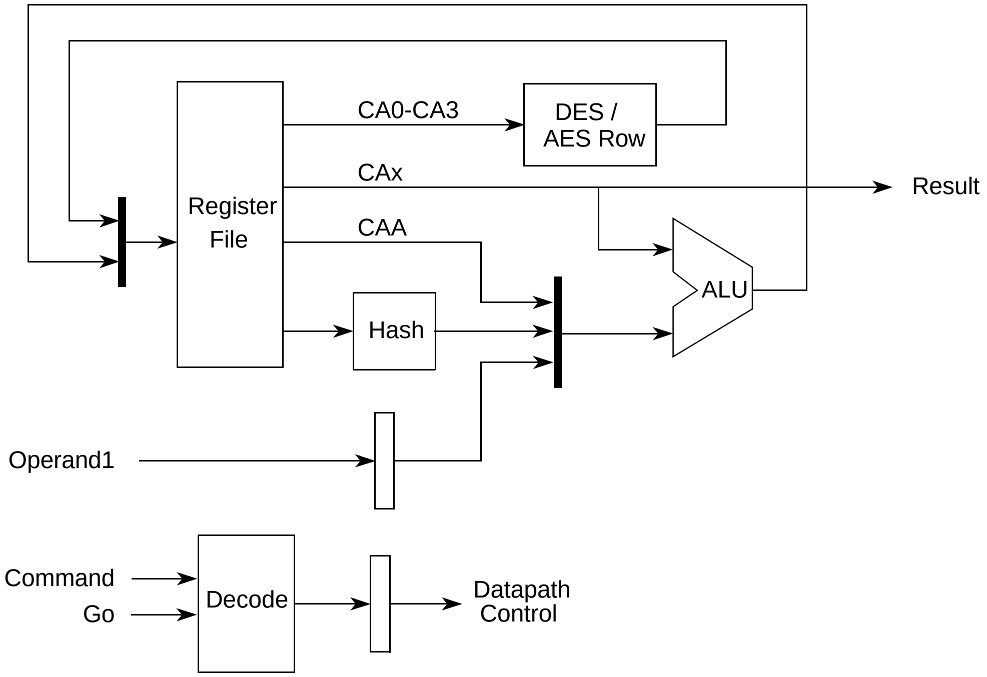


Figure 23-1. Top Level CAU Block Diagram

23.1.2 Overview

The CAU supports acceleration of the following algorithms:

- DES
- 3DES
- AES
- MD5

- SHA-1

This selection of algorithms provides excellent support for network security standards (SSL, IPsec). Additionally, using the CAU efficiently permits the implementation of any higher level functions or modes of operation (HMAC, CBC, etc.) based on the supported algorithm.

The CAU is an instruction-level ColdFire coprocessor. The cryptographic algorithms are implemented partially in software with only functions critical to increasing performance implemented in hardware. The ColdFire coprocessor allows for efficient, fine-grained partitioning of functions between hardware and software.

- Implement the innermost round functions by using the coprocessor instructions
- Implement higher-level functions in software by using the standard ColdFire instructions

This partitioning of functions is key to minimizing size of the CAU while maintaining a high level of throughput. Using software for some functions also simplifies the CAU design. The CAU implements a set of 22 coprocessor commands that operate on a register file of eight 32-bit registers. It is tightly coupled to the ColdFire core and there is no local memory or external interface.

23.1.3 Features

The CAU includes these distinctive features:

- Supports DES, 3DES, AES, MD5, SHA-1 algorithms
- Simple, flexible programming model

23.2 Memory Map/Register Definition

The CAU only supports longword operations and register accesses. All registers support read, write, and ALU operations. However, only bits 1–0 of the CASR are writeable. Bits 31–2 of the CASR must be written as 0 for compatibility with future versions of the CAU.

Table 23-1. CAU Memory Map

Code	Register	DES	AES	SHA-1	MD5	Access	Reset Value	Section/Page
0	CAU status register (CASR)	—	—	—	—	R/W	0x1000_0000	23.2.1/23-3
1	CAU accumulator (CAA)	—	—	T	a	R	0x0000_0000	23.2.2/23-3
2	General purpose register 0 (CA0)	C	W0	A	—	R	0x0000_0000	23.2.3/23-4
3	General purpose register 1 (CA1)	D	W1	B	b	R	0x0000_0000	23.2.3/23-4
4	General purpose register 2 (CA2)	L	W2	C	c	R	0x0000_0000	23.2.3/23-4
5	General purpose register 3 (CA3)	R	W3	D	d	R	0x0000_0000	23.2.3/23-4
6	General purpose register 4 (CA4)	—	—	E	—	R	0x0000_0000	23.2.3/23-4
7	General purpose register 5 (CA5)	—	—	W	—	R	0x0000_0000	23.2.3/23-4

23.2.1 CAU Status Register (CASR)

CASR contains the status and configuration for the CAU.

Register code: 0x0 (CASR)

Access: Read/write via CAU commands

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

Figure 23-2. CAU Status Register (CASR)

Table 23-2. CASR Field Descriptions

Field	Description
31–28 VER	CAU version. Indicates CAU version 0x1 Initial CAU version (This is the value on this device) 0x2 Second version, added support for SHA-256 algorithm
27–2	Reserved, must be cleared.
1 DPE	DES parity error. 0 No error detected 1 DES key parity error detected
0 IC	Illegal command. Indicates an illegal instruction not found in Section 23.3.3, “CAU Commands,” has been executed. 0 No illegal commands issued 1 Illegal coprocessor command issued

23.2.2 CAU Accumulator (CAA)

CAU commands use the CAU accumulator for storage of results and as an operand for the cryptographic algorithms.

Register code: 0x1 (CAA)

Access: Read/write via CAU commands

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	ACC																															
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 23-3. CAU Accumulator Register (CAA)

Table 23-3. CAA Field Descriptions

Field	Description
31–0 ACC	Accumulator. Stores results of various CAU commands.

23.2.3 CAU General Purpose Registers (CA*n*)

The six CAU general purpose registers are used in the CAU commands for storage of results and as operands for the various cryptographic algorithms.

Register code: 0x2 (CA0)
0x3 (CA1)
0x4 (CA2)
0x5 (CA3)
0x6 (CA4)
0x7 (CA5)

Access: Read/write
via CAU commands

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CA*n*																															
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 23-4. CAU General Purpose Registers (CA*n*)

Table 23-4. CA*n* Field Descriptions

Field	Description
31–0 CA*n*	General purpose registers. Used by the CAU commands. Some cryptographic operations work with specific registers.

23.3 Functional Description

23.3.1 Programming Model

The CAU is an instruction-level coprocessor. It has a dedicated register file, a specialized ALU, and specialized units for performing cryptographic operations. The CAU design uses a simple, flexible accumulator-based architecture. Most commands, including load and store, can specify any register in the register file. Some cryptographic operations work with specific registers.

23.3.2 Coprocessor Instructions

Operation of the CAU is controlled via standard ColdFire coprocessor load (cp0ld) and store (cp0st) instructions. The CAU has a dedicated register file accessed using these instructions. The load instruction loads CAU registers and specifies CAU operations. The store instruction stores CAU registers. The example assembler syntax for the CAU is:

```
cp0ld.l         <ea>,<CMD>      ; coprocessor load
cp0st.l         <ea>,<CMD>      ; coprocessor store
```

The <ea> field specifies the source operand (operand1) for load instructions and destination (result) for store instructions. The basic ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)} are supported for this field. The <CMD> field is a 9-bit value that specifies the CAU command for an instruction. Table 23-5 shows how the CAU supports a single command (STR) for store instructions and 21 commands for the load instructions. The CAU only supports longword operations. A CAU command can be issued every clock cycle.

23.3.3 CAU Commands

The CAU supports the commands shown in Table 23-5. All other encodings are reserved. The CASR[IC] bit is set if an undefined command is issued. A specific illegal command (ILL) is defined to allow software self-checking. Reserved commands should not be issued to ensure compatibility with future implementations.

The CMD field specifies the CAU command for the instruction.

Table 23-5. CAU Commands

Inst Type	Command Name	Description	CMD									Operation
			8	7	6	5	4	3	2	1	0	
cp0ld	CNOP	No Operation	0x000									—
cp0ld	LDR	Load Reg	0x01					CAx				Op1 → CAx
cp0st	STR	Store Reg	0x02					CAx				CAx → Result
cp0ld	ADR	Add	0x03					CAx				CAx + Op1 → CAx
cp0ld	RADR	Reverse and Add	0x04					CAx				CAx + ByteRev(Op1) → CAx
cp0ld	ADRA	Add Reg to Acc	0x05					CAx				CAx + CAA → CAA
cp0ld	XOR	Exclusive Or	0x06					CAx				CAx ^ Op1 → CAx
cp0ld	ROTL	Rotate Left	0x07					CAx				CAx <<< Op1 → CAx
cp0ld	MVRA	Move Reg to Acc	0x08					CAx				CAx → CAA
cp0ld	MVAR	Move Acc to Reg	0x09					CAx				CAA → CAx
cp0ld	AESS	AES Sub Bytes	0x0A					CAx				SubBytes(CAx) → CAx
cp0ld	AESIS	AES Inv Sub Bytes	0x0B					CAx				InvSubBytes(CAx) → CAx
cp0ld	AESC	AES Column Op	0x0C					CAx				MixColumns(CAx)^Op1→ CAx
cp0ld	AESIC	AES Inv Column Op	0x0D					CAx				InvMixColumns(CAx^Op1) → CAx
cp0ld	AESR	AES Shift Rows	0x0E0									ShiftRows(CA0-CA3) → CA0-CA3
cp0ld	AESIR	AES Inv Shift Rows	0x0F0									InvShiftRows(CA0-CA3)→ CA0-CA3
cp0ld	DESR	DES Round	0x10					IP	FP	KS[1:0]		DES Round(CA0-CA3)→CA0-CA3
cp0ld	DESK	DES Key Setup	0x11					0	0	CP	DC	DES Key Op(CA0-CA1)→CA0-CA1 Key Parity Error & CP → CASR[1]
cp0ld	HASH	Hash Function	0x12					0	HF[2:0]			Hash Func(CA1-CA3)+CAA→CAA
cp0ld	SHS	Secure Hash Shift	0x130									CAA <<< 5→ CAA, CAA→CA0, CA0→CA1, CA1 <<< 30 → CA2, CA2→CA3, CA3→CA4
cp0ld	MDS	Message Digest Shift	0x140									CA3→CAA, CAA→CA1, CA1→CA2, CA2→CA3,
cp0ld	ILL	Illegal Command	0x1F0									0x1→CASR[0]

Section 23.4.2, “Assembler Equate Values,” contains a set of assembly constants used in the command descriptions here. If supported by the assembler, macros can also be created for each instruction. The value CA*x* should be interpreted as any CAU register (CASR, CAA, CA*n*) and the <ea> field as one of the supported ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)}. For example, the instruction to add the value from the core register D1 to the CAU register CA0 is:

```
cp0ld.l         %d1,#ADR+CA0        ; CA0=CA0+d1
```

23.3.3.1 Coprocessor No Operation (CNOP)

```
cp0ld.l   #CNOP
```

The CNOP command is the coprocessor no-op defined by the ColdFire coprocessor definition for synchronization. It is not actually issued to the coprocessor from the core.

23.3.3.2 Load Register (LDR)

```
cp0ld.l   <ea>,#LDR+CAx
```

The LDR command loads CAx with the source data specified by <ea>.

23.3.3.3 Store Register (STR)

```
cp0st.l   <ea>,#STR+CAx
```

The STR command stores the value from CAx to the destination specified by <ea>.

23.3.3.4 Add to Register (ADR)

```
cp0ld.l   <ea>,#ADR+CAx
```

The ADR command adds the source operand specified by <ea> to CAx and stores the result in CAx.

23.3.3.5 Reverse and Add to Register (RADR)

```
cp0ld.l   <ea>,#RADR+CAx
```

The RADR command performs a byte reverse on the source operand specified by <ea>, adds that value to CAx, and stores the result in CAx. Table 23-6 shows an example.

Table 23-6. RADR Command Example

Operand	CAx Before	CAx After
0x0102_0304	0xA0B0_C0D0	0xA4B3_C2D1

23.3.3.6 Add Register to Accumulator (ADRA)

```
cp0ld.l   #ADRA+CAx
```

The ADRA command adds CAx to CAA and stores the result in CAA.

23.3.3.7 Exclusive Or (XOR)

```
cp0ld.l   <ea>,#XOR+CAx
```

The XOR command performs an exclusive-or of the source operand specified by <ea> with CAx and stores the result in CAx.

23.3.3.8 Rotate Left (ROTL)

```
cp0ld.l    <ea>,#ROTL+CAx
```

ROTL rotates the CAx bits to the left with the result stored back to CAx. The number of bits to rotate is the value specified by <ea> modulo 32.

23.3.3.9 Move Register to Accumulator (MVRA)

```
cp0ld.l    #MVRA+CAx
```

The MVRA command moves the value from the source register CAx to the destination register CAA.

23.3.3.10 Move Accumulator to Register (MVAR)

```
cp0ld.l    #MVAR+CAx
```

The MVAR command moves the value from source register CAA to the destination register CAx.

23.3.3.11 AES Substitution (AESS)

```
cp0ld.l    #AESS+CAx
```

The AESS command performs the AES byte substitution operation on CAx and stores the result back to CAx.

23.3.3.12 AES Inverse Substitution (AESIS)

```
cp0ld.l    #AESIS+CAx
```

The AESIS command performs the AES inverse byte substitution operation on CAx and stores the result back to CAx.

23.3.3.13 AES Column Operation (AESC)

```
cp0ld.l    <ea>,#AESC+CAx
```

The AESC command performs the AES column operation on the contents of CAx then performs an exclusive-or of that result with the source operand specified by <ea> and stores the result in CAx.

23.3.3.14 AES Inverse Column Operation (AESIC)

```
cp0ld.l    <ea>,#AESIC+CAx
```

The AESIC command performs an exclusive-or operation of the source operand specified by <ea> on the contents of CAx followed by the AES inverse mix column operation on that result and stores the result back in CAx.

23.3.3.15 AES Shift Rows (AESR)

```
cp0ld.l    #AESR
```

The AESR command performs the AES shift rows operation on registers CA0, CA1, CA2, and CA3. Table 23-7 shows an example.

Table 23-7. AESR Command Example

Register	Before	After
CA0	0x0102_0304	0x0106_0B00
CA1	0x0506_0708	0x050A_0F04
CA2	0x090A_0B0C	0x090E_0308
CA3	0x0D0E_0F00	0x0D02_070C

23.3.3.16 AES Inverse Shift Rows (AESIR)

```
cp0ld.l    #AESIR
```

The AESIR command performs the AES inverse shift rows operation on registers CA0, CA1, CA2 and CA3. Table 23-8 has an example.

Table 23-8. AESIR Command Example

Register	Before	After
CA0	01060B00	01020304
CA1	050A0F04	05060708
CA2	090E0308	090A0B0C
CA3	0D02070C	0D0E0F00

23.3.3.17 DES Round (DESR)

```
cp0ld.l    #DESR+{IP}+{FP}+{KSx}
```

The DESR command performs a round of the DES algorithm and a key schedule update with the following source and destination designations: CA0=C, CA1=D, CA2=L, CA3=R. If the IP bit is set, DES initial permutation performs on CA2 and CA3 before the round operation. If the FP bit is set, DES final permutation (inverse initial permutation) performs on CA2 and CA3 after the round operation. The round operation uses the source values from registers CA0 and CA1 for the key addition operation. The KSx field specifies the shift for the key schedule operation to update the values in CA0 and CA1. Table 23-9 defines the specific shift function performed based on the KSx field.

Table 23-9. Key Shift Function Codes

KSx Code	KSx Define	Shift Function
0	KSL1	Left 1
1	KSL2	Left 2

Table 23-9. Key Shift Function Codes (continued)

KSx Code	KSx Define	Shift Function
2	KSR1	Right 1
3	KSR2	Right 2

23.3.3.18 DES Key Setup (DESK)

```
        cp0ld.l    #DESK+{CP}+{DC}
```

The DESK command performs the initial key transformation (permuted choice 1) defined by the DES algorithm on CA0 and CA1 with CA0 containing bits 1–32 of the key and CA1 containing bits 33–64 of the key[1]. If the DC bit is set, no shift operation performs and the values C_0 and D_0 store back to CA0 and CA1 respectively. The DC bit should be set for decrypt operations. If the DC bit is not set, a left shift by one also occurs and the values C_1 and D_1 store back to CA0 and CA1 respectively. The DC bit should be cleared for encrypt operations. If the CP bit is set and a key parity error is detected, CASR[DPE] bit is set; otherwise, it is cleared.

23.3.3.19 Hash Function (HASH)

```
        cp0ld.l    #HASH+HFx
```

The HASH command performs a hashing operation on a set of registers and adds that result to the value in CAA and stores the result in CAA. The specific hash function performed is based on the HFx field as defined in Table 23-10.

Table 23-10. Hash Function Codes

HFx Code	HFx Define	Hash Function	Hash Logic
0	HFF	MD5 F()	(CA1 & CA2) | ($\overline{\text{CA1}}$ & CA3)
1	HFG	MD5 G()	(CA1 & CA3) | (CA2 & $\overline{\text{CA3}}$)
2	HFH	MD5 H(), SHA Parity()	CA1 ^ CA2 ^ CA3
3	HFI	MD5 I()	CA2 ^ (CA1 | $\overline{\text{CA3}}$)
4	HFC	SHA Ch()	(CA1 & CA2) ^ ($\overline{\text{CA1}}$ & CA3)
5	HFM	SHA Maj()	(CA1 & CA2) ^ (CA1 & CA3) ^ (CA2 & CA3)

23.3.3.20 Secure Hash Shift (SHS)

```
        cp0ld.l    #SHS
```

The SHS command does a set of parallel register-to-register move and shift operations for implementing SHA-1. The following source and destination assignments are made: CAA=CAA<<<5, CA0=CAA, CA1=CA0, CA2=CA1<<<30, CA3=CA2, CA4=CA3.

[1] The DES algorithm numbers the most significant bit of a block as bit 1 and the least significant as bit 64.

23.3.3.21 Message Digest Shift (MDS)

```
      cp0ld.l   #MDS
```

The MDS command does a set of parallel register-to-register move operations for implementing MD5. The following source and destination assignments are made: CAA=CA3, CA1=CAA, CA2=CA1, CA3=CA2.

23.3.3.22 Illegal Command (ILL)

```
      cp0ld.l   #ILL
```

The ILL command is a specific illegal command that sets CASR[IC]. All other illegal commands are reserved for use in future implementations.

23.4 Application/Initialization Information

23.4.1 Code Example

A code fragment is shown below as an example of how the CAU is used. This example shows the round function of the AES algorithm. Core register A0 is pointing to the key schedule.

```
      cp0ld.l   #AESS+CA0          ; sub bytes w0
      cp0ld.l   #AESS+CA1          ; sub bytes w1
      cp0ld.l   #AESS+CA2          ; sub bytes w2
      cp0ld.l   #AESS+CA3          ; sub bytes w3
      cp0ld.l   #AESR              ; shift rows
      cp0ld.l   (%a0)+,#AESC+CA0  ; mix col, add key w0
      cp0ld.l   (%a0)+,#AESC+CA1  ; mix col, add key w1
      cp0ld.l   (%a0)+,#AESC+CA2  ; mix col, add key w2
      cp0ld.l   (%a0)+,#AESC+CA3  ; mix col, add key w3
```

23.4.2 Assembler Equate Values

The following equates ease programming of the CAU.

```
; CAU Registers (CAx)
         .set     CASR,0x0
         .set     CAA,0x1
         .set     CA0,0x2
         .set     CA1,0x3
         .set     CA2,0x4
         .set     CA3,0x5
         .set     CA4,0x6
         .set     CA5,0x7

; CAU Commands
         .set     CNOP,0x000
         .set     LDR,0x010
         .set     STR,0x020
         .set     ADR,0x030
         .set     RADR,0x040
         .set     ADRA,0x050
         .set     XOR,0x060
         .set     ROTL,0x070
         .set     MVRA,0x080
         .set     MVAR,0x090
         .set     AESS,0x0A0
         .set     AESIS,0x0B0
         .set     AESC,0x0C0
         .set     AESIC,0x0D0
         .set     AESR,0x0E0
         .set     AESIR,0x0F0
         .set     DESR,0x100
         .set     DESK,0x110
         .set     HASH,0x120
         .set     SHS,0x130
         .set     MDS,0x140
         .set     ILL,0x1F0

; DESR  Fields
         .set     IP,0x08            ; initial permutation
         .set     FP,0x04            ; final permutation
         .set     KSL1,0x00          ; key schedule left 1 bit
         .set     KSL2,0x01          ; key schedule left 2 bits
         .set     KSR1,0x02          ; key schedule right 1 bit
         .set     KSR2,0x03          ; key schedule right 2 bits

; DESK Field
         .set     DC,0x01            ; decrypt key schedule
         .set     CP,0x02            ; check parity

; HASH Functions Codes
         .set     HFF,0x0            ; MD5 F() CA1&CA2 | ~CA1&CA3
         .set     HFG,0x1            ; MD5 G() CA1&CA3 | CA2&~CA3
         .set     HFH,0x2            ; MD5 H(), SHA Parity() CA1^CA2^CA3
         .set     HFI,0x3            ; MD5 I()  CA2^(CA1|~CA3)
         .set     HFC,0x4            ; SHA Ch() CA1&CA2 ^ ~CA1&CA3
         .set     HFM,0x5            ; SHA Maj() CA1&CA2 ^ CA1&CA3 ^ CA2&CA3
```

Chapter 24
Random Number Generator (RNG)

24.1 Introduction

This chapter describes the random number generator (RNG), including a programming model, functional description, and application information.

NOTE

The MCF53010 and MCF53012 do not contain cryptography modules. Refer to Table 1-1 for details on device configurations.

24.1.1 Block Diagram

Figure 24-1 shows the RNG’s three main blocks: PRNG, TRNG, and XSEED generator. Section 24.4, “Functional Description,” describes these blocks in more detail.

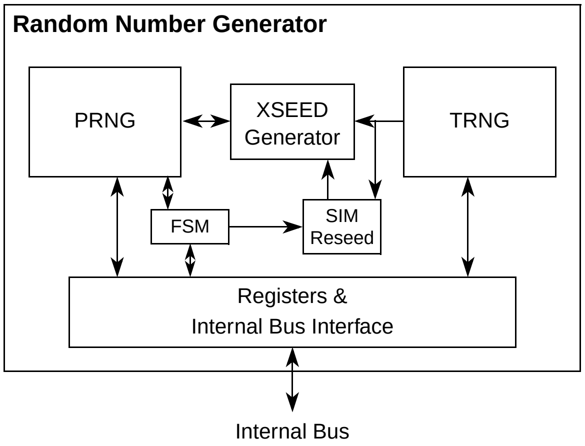


Figure 24-1. RNG Block Diagram

24.1.2 Overview

The purpose of the RNG is to generate cryptographically strong random data. It uses a true random number generator (TRNG) and a pseudo-random number generator (PRNG) to achieve true randomness and cryptographic strength. The RNG generates random numbers for secret keys, per message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

24.1.3 Features

The RNG includes these distinctive features:

- National Institute of Standards and Technology (NIST)-approved pseudo-random number generator
  - — http://csrc.nist.gov
- Supports the key generation algorithm defined in the Digital Signature Standard
  - — http://www.itl.nist.gov/fipspubs/fip186.htm
- Integrated entropy sources capable of providing the PRNG with entropy for its seed.

24.2 Modes of Operation

The RNG operates in the following modes.

24.2.1 Self Test Mode

In this mode the RNG performs a self test of the statistical counters and the PRNG algorithm to verify that the hardware is functioning properly. The self test takes ~29,000 cycles to complete. When self test completes an interrupt may be generated, if there are no outstanding commands in the command register. This mode is entered by setting the RNGCMD[ST] bit. When self test mode completes, the RNG remains idle until seed mode is requested or the RNG transitions to seed mode if automatic seeding is enabled.

24.2.2 Seed Generation Mode

During seed generation, the RNG adds entropy generated in the TRNG to the 256-bit XKEY register. The PRNG algorithm executes 20,000 times sampling the entropy from the TRNG to create an initial seed for random number generation. At the same time, the TRNG runs simple statistical tests on its output.

When seed generation is complete, the TRNG reports the pass/fail result of the tests through RNGESR. If the new seed passes the statistical tests, RNGSR[SDN] is set, signalling that the RNG is ready to compute secure pseudo-random data. The RNG then transitions to random number generation mode.

24.2.3 Random Number Generation Mode

When seed generation mode completes and the output FIFO is empty, the RNG enters this mode automatically. Random number generation mode quickly creates computationally random data that is derived by the initial seed produced in seed generation mode.

During random number generation, a new 160-bit random number is generated whenever the five word output FIFO is empty. When the output FIFO contains data, the RNG automatically enters sleep mode, waiting for the data to be read. When the data is read, the RNG generates a new 160-bit word and goes back to sleep.

After generating 2^{20} words of random data, the RNG lets the user know that it requires reseeding through RNGSR and continues to generate random data until it is directed to reseed. However, if auto-seeding is selected, the RNG automatically completes seeding whenever it is needed.

24.3 Memory Map/Register Definition

Table 24-1 shows the address map for the RNG module. Detailed register descriptions are found in the following section.

Table 24-1. RNG Block Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_4000	RNG Version ID Register (RNGVER)	32	R/W	0x1000_0280	24.3.1/24-3
0xFC0C_4004	RNG Command Register (RNGCMD)	32	R/W	0x0000_0000	24.3.2/24-4
0xFC0C_4008	RNG Control Register (RNGCR)	32	R/W	0x0000_0000	24.3.3/24-5
0xFC0C_400C	RNG Status Register (RNGSR)	32	R	0x0000_500D	24.3.4/24-6
0xFC0C_4010	RNG Error Status Register (RNGESR)	32	R	0x0000_0000	24.3.5/24-7
0xFC0C_4014	RNG Output FIFO (RNGOUT)	32	R	0x0000_0000	24.3.6/24-8
0xFC0C_4018	RNG Entropy Register (RNGER)	32	W	0x0000_0000	24.3.7/24-8

24.3.1 RNG Version ID Register (RNGVER)

The read-only RNGVER register contains the current version of the RNG. It consists of the RNG type and major and minor revision numbers.

Address: 0xFC0C_4000 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	TYPE				0	0	0	0	0	0	0	0	0	0	0	0	MAJOR								MINOR							
W																																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0

Figure 24-2. RNG Version Register (RNGVER)

Table 24-2. RNGVER Field Descriptions

Field	Description
31–28 TYPE	Random number generator type. 0000 RNGA 0001 RNGB (This is the type used in this module) 0010 RNGC Else Reserved
27–16	Reserved, must be cleared.
15–8 MAJOR	Major version number. This field is always set to 0x02.
7–0 MINOR	Minor version number. Subject to change.

24.3.2 RNG Command Register (RNGCMD)

RNGCMD controls the RNG’s operating modes and interrupt status.

Address: 0xFC0C_4004 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	GS	ST
W																										SR	CE	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 24-3. RNG Command Register (RNGCMD)

Table 24-3. RNGCMD Field Descriptions

Field	Description
31–7	Reserved, must be cleared.
6 SR	Software reset. Performs a software reset of the RNG. This bit is self-clearing. 0 Do not perform a software reset 1 Software reset
5 CE	Clear error. Clears the errors in the RNGESR register and the RNG interrupt. This bit is self-clearing. 0 Do not clear errors and interrupt 1 Clear errors and interrupt
4 CI	Clear interrupt. Clears the RNG interrupt if an error is not present. This bit is self-clearing. 0 Do not clear interrupt 1 Clear interrupt
3–2	Reserved, must be cleared.
1 GS	Generate seed. Initiates the seed generation process described in Section 24.2.2, “Seed Generation Mode”. Seed generation starts • When RNGSR[BUSY] is cleared • If set simultaneously with ST, after self-test When the seed generation process completes, this bit automatically clears and an interrupt may be generated if all requested operations are complete. 0 Not in seed generation mode 1 Generate seed mode
0 ST	Self test. Initiates a self test of the RNG’s internal logic, described in Section 24.2.1, “Self Test Mode”. The self-test starts • When RNGSR[BUSY] is cleared, or • If set simultaneously with GS, self test takes precedence and is completed first. When self test completes, this bit automatically clears and an interrupt may be generated if all requested operations are complete. 0 Not in self test mode 1 Self test mode

24.3.3 RNG Control Register (RNGCR)

Through use of this register, the RNG can be programmed to provide slightly different functionality based on its desired use.

Address: 0xFC0C_4008 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	MSK ERR	MSK DN	AS	0	0	FUF MOD	
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 24-4. RNG Control Register (RNGCR)

Table 24-4. RNGCR Field Descriptions

Field	Description
31–7	Reserved, must be cleared.
7	Reserved, must be cleared.
6 MSKERR	Mask error interrupt. Masks interrupts generated by errors in the RNG. These errors can still be viewed in RNGESR. 0 No mask applied 1 Mask applied to the error interrupt
5 MSKDN	Mask done interrupt. Masks interrupts generated upon completion of seed and self test modes. The status of these jobs can be viewed by: • Reading RNGSR and viewing the seed done and self test done bits (RNGSR[SDN, STDN]) • Viewing RNGCMD for generate seed or self test bits (RNGCMD[GS,ST]) being set, indicating that the operation is still taking place. 0 No mask applied 1 Mask applied
4 AR	Auto-reseed. Setting this bit allows the RNG to automatically generate a new seed whenever one is needed. This allows software to never use the RNGCMD[GS], although it is still possible. A new seed is needed whenever the RNGSR[RS] is set. 0 Do not enable automatic reseeding 1 Enable automatic reseeding
3–2	Reserved, must be cleared.
1–0 FUFMOD	FIFO underflow response mode. Controls the RNG's response to a FIFO underflow condition. 00 Return all zeros and set RNGESR[FUF] 01 Generate bus transfer wait until data is available, then return generated word 10 Generate bus transfer error. See Chapter 11, "System Control Module (SCM)", for more details. 11 Generate interrupt and return all zeros. (Overrides RNGCR[MSKERR].)

24.3.4 RNG Status Register (RNGSR)

The RNGSR is a read-only register which reflects the internal status of the RNG.

Address: 0xFC0C_400C Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STATPF								SELFPF		0	0	0	0	0	ERR
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	FS				FL				0	NSDN	SDN	STDN	RS	SLP	BUSY	1
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1

Figure 24-5. RNG Status Register (RNGSR)

Table 24-5. RNGSR Field Descriptions

Field	Description
31–24 STATPF	Statistics test pass fail. Indicates pass or fail status of the various statistics tests on the last seed generated. • Bit 31 - Long run test (>34) • Bit 30 - Length 6+ run test • Bit 29 - Length 5 run test • Bit 28 - Length 4 run test • Bit 27 - Length 3 run test • Bit 26 - Length 2 run test • Bit 25 - Length 1 run test • Bit 24 - Monobit test 0 Pass 1 Fail
23–22 SELFPF	Self Test Pass Fail. Indicates Pass or Fail status of the TRNG and PRNG Self Tests, 1 indicates failure, 0 pass. • Bit 23 - TRNG self test pass/fail • Bit 22 - PRNG self test pass/fail 0 Pass 1 Fail
21–17	Reserved, must be cleared.
16 ERR	Error. Indicates an error was detected in the RNG. Read the RNGESR register for details. 0 No error 1 Error detected
15–12 FS	FIFO size. Size of the FIFO, and maximum possible FIFO level. This value is set to five.
11–8 FL	FIFO level. Indicates the number of random words currently in the output FIFO.
6 NSDN	New seed done. Indicates that a new seed is ready for use during the next seed generation process.
5 SDN	Seed done. Indicates the RNG has generated the first seed. 0 Seed generation process not complete 1 Completed seed generation since the last reset

Table 24-5. RNGSR Field Descriptions (continued)

Field	Description
4 STDN	Self test done. Indicates the self test is complete. This bit is cleared by hardware reset or a new self test is initiated by setting RNGCMD[ST]. 0 Self test not complete 1 Completed a self test since the last reset
3 RS	Reseed needed. Indicates the RNG needs to be reseeded. This is done by setting RNGCMD[GS], or automatically if RNGCR[AR] is set. 0 RNG does not need to be reseeded 1 RNG needs to be reseeded
2 SLP	Sleep. Indicates if the RNG is in sleep mode. When set, the RNG is in sleep mode and all internal clocks are disabled. While in this mode, access to the FIFO is allowed. Once the FIFO is empty, the RNG fills the FIFO and then enters sleep mode again. 0 RNG is not in sleep mode 1 RNG is in sleep mode
1 BUSY	Busy. Reflects the current state of RNG. If RNG is currently seeding, generating the next seed, creating a new random number, or performing a self test, this bit is set. 0 Not busy 1 Busy
0	Reserved, must be set.

24.3.5 RNG Error Status Register (RNGESR)

Address: 0xFC0C_4010 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	FUF	SATE	STE	OSCE	LFE
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 24-6. RNG Error Status Register (RNGESR)

Table 24-6. RNGESR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 FUF	FIFO underflow error. Indicates the RNG has experienced a FIFO underflow condition resulting in the last random data read being unreliable. This bit can be masked by RNGCR[FUFMOD] and is cleared by hard or soft reset or by setting RNGCMD[CE]. 1 FIFO underflow has occurred 0 FIFO underflow has not occurred
3 SATE	Statistical test error. Indicates if RNG has failed the statistical tests for the last generated seed. This bit is sticky and is cleared by a hardware or software reset, setting RNGCMD[CE]. 1 RNG has failed the statistical tests during initialization 0 RNG has not failed the statistical tests
2 STE	Self test error. Indicates the RNG has failed the most recent self test. This bit is sticky and can only be reset by a software or hardware reset. 0 RNG has not failed self test 1 RNG has failed self test

Table 24-6. RNGESR Field Descriptions (continued)

Field	Description
1 OSCE	Oscillator error. Indicates the oscillator in the RNG may be broken. This bit is sticky and can only be cleared by a software or hardware reset. 0 RNG oscillator is working properly 1 Problem detected with the RNG oscillator
0 LFE	Linear feedback shift register (LFSR) error. When this bit is set, the interrupt generated was caused by a failure of one of the LFSRs in one of the RNG's three entropy sources. This bit is sticky and can only be reset via a hardware reset. 0 LFSRs are working properly 1 LFSR failure has occurred

24.3.6 RNG Output FIFO (RNGOUT)

The RNGOUT provides temporary storage for random data generated by the RNG. This allows the user to read multiple random longwords back-to-back. A read of this address when the FIFO is not empty, returns 32 bits of random data. If the FIFO is read when empty, a FIFO underrun response is returned according to RNGCR[FUFMOD]. For optimal system performance, poll RNGSR[FL] to ensure random values are present before reading the FIFO.

Address: 0xFC0C_4014 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 24-7. RNG Output FIFO (RNGOUT)

24.3.7 RNG Entropy Register (RNGER)

The RNGER is a write-only register which allows the user to insert entropy into the RNG. When written this register causes an addition of the write data to the XKEY within the RNG. If the system has a means to collect quality entropy (user key strokes or other random patterns), this is the way to add it directly into the accumulated entropy within the RNG. Writing the RNGER does not have a detrimental effect on the quality of random numbers as the number written is added to the state, not used to replace the existing state. Writing all zeros to the entropy register does nothing to the quality of random numbers generated.

NOTE

This register can only be written when the RNG is not busy (RNGSR[BUSY] = 0).

Address: 0xFC0C_4018 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	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 24-8. RNG Entropy Register (RNGER)

Table 24-7. RNGER Field Descriptions

Field	Description
31–0 ENT	Entropy input. This value is added directly to the internal state of the RNG thus modifying its internal state and affecting future random numbers generated by the RNG.

24.4 Functional Description

The RNG performs two functional operations, as described in Section 24.2, “Modes of Operation”: seed generation and random number generation. Theses operations are performed with cooperation from the two major functional blocks in the RNG described below.

24.4.1 Pseudorandom Number Generator (PRNG)

The PRNG implements the NIST-approved PRNG described in the *Digital Signature Standard*. The 160-bit output of the SHA-1 block is the next five words of random data. The PRNG is designed to generate 2^{20} words of random data before requiring reseeding, using the TRNG only during the seeding/initialization process. The initial seed takes approximately two million clock cycles. After this the RNG can generate five 32-bit words every 112 clock cycles. Reseeding takes place transparently through use of the simultaneous reseed LFSRs. The entropy stored in this 128-bit LFSR and 128-bit shift register is added directly into the XKEY structure via the RNG XSEED generator whenever reseeding is required.

24.4.2 True Random Number Generator (TRNG)

The TRNG is comprised of two entropy sources each providing a single bit of output. Concatenated together, these two output bits are expected to provide one bit of entropy every 100 clock cycles. In addition to generating entropy, the TRNG also performs several statistical tests on its output. The pass/fail status of these tests are reflected in RNGESR.

24.4.3 RNG Interrupts

There is a single RNG interrupt generated to the processor’s interrupt controller. The source of the interrupt is determined by reading the RNG status register. If an error is the cause of the interrupt, further information is available by reading the RNG error status register. The available sources are described in the following table.

Table 24-8. RNG Interrupt Sources

Sources	Status Bit Field	Mask Bit Field	Description
Error	RNGSR[ERR]	RNGCR[MERR]	Error detected. See RNGESR for details.
LFSR	RNGESR[LSFRE]	RNGCR[MERR]	Fault in one of the TRNG’s LFSRs
Oscillator	RNGESR[OSCE]	RNGCR[MERR]	TRNG ring oscillator may be malfunctioning
Self test	RNGESR[STE]	RNGCR[MERR]	Self test failed

Table 24-8. RNG Interrupt Sources (continued)

Sources		Status Bit Field	Mask Bit Field	Description
	Statistical test	RNGESR[SATE]	RNGCR[MERR]	Statistics test for last seed generation failed
	FIFO Underflow	RNGESR[FUF]	RNGCR[MERR]	FIFO read while empty
Seed generation done		RNGSR[SDN]	RNGCR[MSKDN]	First seed was generated
Self test done		RNGSR[STDN]	RNGCR[MSKDN]	Self test finished

24.5 Initialization/Application Information

24.5.1 Manual Seeding

The intended general operation of the RNG is as follows:

1. Reset/initialize.
2. Write to the RNG control register to setup the RNG for the desired functionality.
3. Write to RNGCMD to run self-test or seed generation.
4. Wait for interrupt to indicate completion of the requested operation(s).
5. Repeat steps 3–4 if seed generation is not complete.
6. Poll RNGSR for FIFO level.
7. Read available random data from output FIFO.
8. Repeat steps 6 and 7 as needed, until 2^{20} words have been generated.
9. Write to RNGCMD to run seed mode.
10. Repeat steps 4–9.

24.5.2 Automatic Seeding

The intended general operation of the RNG with automatic seeding enabled is as follows:

1. Reset/initialize.
2. Write to the RNG control register to setup the RNG for automatic seeding and the desired functionality.
3. Wait for interrupt to indicate completion of first seed
4. Poll RNGSR for FIFO level.
5. Read available random data from output FIFO.
6. Repeat steps 4 and 5 as needed. Automatic seeding occurs when necessary and is transparent to operation.

Chapter 25
IC Identification (IIM)

25.1 Introduction

The IIM provides an interface for reading and programming production information, as well as providing an electrically fusable method for unique device keying.

25.2 Overview

The IIM provides the primary mechanism for interfacing with on-chip fuse elements. Among the uses for the fuses are unique chip identifiers, mask revision numbers, cryptographic keys, and various control signals requiring permanent non-volatility.

As shown in Figure 25-1, the IIM consists of a master controller, a software fuse value shadow cache, and a set of registers to hold the values of signals visible outside the module.

The e-fuses may be blown under software control at the customer factory or in the field. They include a mechanism to inhibit further blowing of fuses (write-protect) to support secure computing environments. The fuse values may also be overridden by software without modifying the fuse element. Similar to the write-protect functionality, the override functionality can also be permanently disabled.

The fuses are divided into banks, with specific intended uses assigned to each bank. Recommended bit usage is shown for several fuse registers. Following the recommended bit usage maximizes software reuse.

25.2.1 Block Diagram

See Figure 25-1 for illustration of the IIM micro architecture.

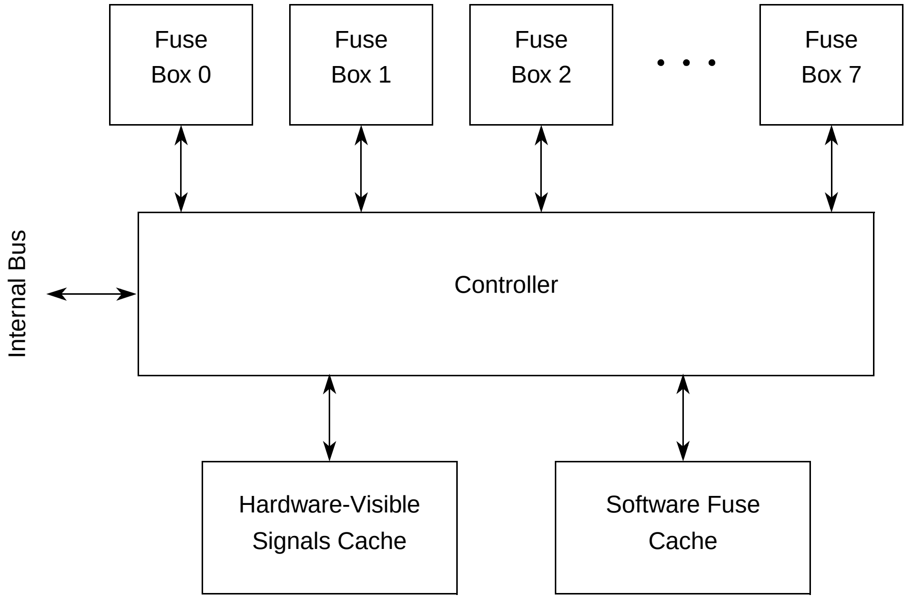


Figure 25-1. IIM Block Diagram

25.2.2 Features

- Up to eight independent fuse banks
  - Fuse banks 0–1 reserved for hardware device configuration
  - Fuse banks 2–7 available for customer use via software-programming
- Maximum usable fuse bank size is 2048 bits total
- Ability to write-protect e-fuses on a per-bank basis
-

25.2.3 Modes of Operation

The IIM is in its functional mode (all specified functionality available) any time it is out of reset and supplied with the proper clocks.

25.3 External Signal Description

The IIM has no external signals.

25.4 Memory Map/Register Definition

All registers are 8-bit wide, but addressable on 32-bit boundaries. The top 24 bits always read as 0 and writes to them are ignored. Table 25-1 shows the IIM memory map.

Table 25-1. IIM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_8000	Status register (IIM_SR)	8	R/W	See Section	25.4.1/25-4
0xFC0C_8004	Status interrupt mask register (IIM_SIMR)	8	R/W	0x0000_0000	25.4.2/25-5
0xFC0C_8008	Error status register (IIM_ESR)	8	R/W	See Section	25.4.3/25-5
0xFC0C_800C	Error Interrupt Mask register (IIM_EIMR)	8	R/W	0x0000_0000	25.4.4/25-6
0xFC0C_8010	Fuse control register (IIM_FCR)	8	R/W	0x0000_0030	25.4.5/25-7
0xFC0C_8014	Upper address register (IIM_UA)	8	R/W	0x0000_0000	25.4.6/25-8
0xFC0C_8018	Lower address register (IIM_LA)	8	R/W	0x0000_0000	25.4.7/25-9
0xFC0C_801C	Explicit sense data register (IIM_SDAT)	8	R	0x0000_0000	25.4.8/25-9
0xFC0C_8028	Program protection register (IIM_PRGP)	8	R/W	0x0000_0000	25.4.9/25-10
0xFC0C_803C	Divide factor register (IIM_DIVIDE)	8	R/W	0x0000_0042	25.4.10/25-10
0xFC0C_8800	Fuse bank 0 protection register (IIM_FBAC0)	8	R/W	See Section	25.4.11.1/25-11
0xFC0C_8804	Analog test select for codec (IIM_ANATEST)	8	R/W	See Section	25.4.11.2/25-12
0xFC0C_8808	Bandgap trim select for codec (IIM_BGPTRIM)	8	R/W	See Section	25.4.11.3/25-12
0xFC0C_880C	Bypass control for codec amplifiers (IIM_AMPBR)	8	R/W	See Section	25.4.11.4/25-13
0xFC0C_8810	Driver amps control for termal limit and alternate common mode (IIM_DRVACR)	8	R/W	See Section	25.4.11.5/25-13
0xFC0C_8814	Speaker amplifiers control (IIM_SPKACR)	8	R/W	See Section	25.4.11.6/25-14
0xFC0C_8818	Handset and headphone control (IIM_HSHPACR)	8	R/W	See Section	25.4.11.7/25-15
0xFC0C_881C ... 0xFC0C_886C	168-bit secure key for security controller (IIM_ SECKEY*n*) *n* = 0:20	8	R/W	See Section	25.4.11.8/25-15
0xFC0C_8874	Self adjustment register for KRAM (IIM_SAKRAM)	8	R/W	See Section	25.4.11.9/25-16
0xFC0C_8878	Self adjustment register for TAG (IIM_SATAG)	8	R/W	See Section	25.4.11.10/25-16
0xFC0C_887C	Self adjustment register for cache (IIM_SACACHE)	8	R/W	See Section	25.4.11.11/25-17
0xFC0C_8C00	Fuse bank 1 protection register (IIM_FBAC1)	8	R/W	See Section	25.4.11.12/25-17
0xFC0C_8C04	Self adjustment register for SCCM (IIM_SASCCM)	8	R/W	See Section	25.4.11.13/25-18
0xFC0C_8C08	Device ID (IIM_DEVID)	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_8C0C ... 0xFC0C_8C7C	Fuse bank 1 data (IIM_FB1W*n*) *n* = 3–31	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_9000	Fuse bank 2 protection register (IIM_FBAC2)	8	R/W	See Section	25.4.11.1/25-11
0xFC0C_9004 ... 0xFC0C_907C	Fuse bank 2 data (IIM_FB2W*n*) *n* = 1–31	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_9400	Fuse bank 3 protection register (IIM_FBAC3)	8	R/W	See Section	25.4.11.1/25-11

Table 25-1. IIM Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_9404 ... 0xFC0C_947C	Fuse bank 3 data (IIM_FB3W*n*) *n* = 1–31	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_9800	Fuse bank 4 protection register (IIM_FBAC4)	8	R/W	See Section	25.4.11.1/25-11
0xFC0C_9804 ... 0xFC0C_987C	Fuse bank 4 data (IIM_FB4W*n*) *n* = 1–31	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_9C00	Fuse bank 5 protection register (IIM_FBAC5)	8	R/W	See Section	25.4.11.1/25-11
0xFC0C_9C04 ... 0xFC0C_9C7C	Fuse bank 5 data (IIM_FB5W*n*) *n* = 1–31	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_A000	Fuse bank 6 protection register (IIM_FBAC6)	8	R/W	See Section	25.4.11.1/25-11
0xFC0C_A004 ... 0xFC0C_A07C	Fuse bank 6 data (IIM_FB6W*n*) *n* = 1–31	8	R/W	See Section	25.4.11.14/25-18
0xFC0C_A400	Fuse bank 7 protection register (IIM_FBAC7)	8	R/W	See Section	25.4.11.1/25-11
0xFC0C_A404 ... 0xFC0C_A47C	Fuse bank 7 data (IIM_FB7W*n*) *n* = 1–31	8	R/W	See Section	25.4.11.14/25-18

25.4.1 IIM Status Register (IIM_SR)

IIM_SR contains all module status information.

Address: 0xFC0C_8000 (IIM_SR)

Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R	BUSY	0					PRGD	SNSD
W							w1c	w1c
Reset	0	0	—	—	—	—	—	0

Figure 25-2. IIM Status Register (IIM_SR)

Table 25-2. IIM_SR Field Descriptions

Field	Description
7 BUSY	Indicates whether the IIM is busy with a program or sense cycle. Any attempt to access the IIM registers other than IIM_SR while it is busy with a program or sense cycle (BUSY set) results in a bus error. 0 The IIM is not busy with a program or sense cycle 1 The IIM is busy with a program or sense cycle
6–2	Reserved, must be cleared.

Table 25-2. IIM_SR Field Descriptions (continued)

Field	Description
1 PRGD	Program done. Indicates an e-fuse program operation is complete. When set an interrupt request is generated if IIM_SIMR[PRGDM] is set. This bit is automatically set by hardware upon completion of an e-fuse program cycle; software must clear the bit by writing 1 to it. 0 Program operation has not finished (read); no meaning (write) 1 Program operation has finished (read); clear bit (write)
0 SNSD	Explicit sense cycle done. Indicates an explicit fuse sense cycle is done, and the data is available in IIM_SDAT. When this bit is set an interrupt request is generated if IIM_SIMR[SNSDM] is set. This bit is automatically set by hardware and must be cleared by writing 1 to it. 0 No explicit sense cycle has finished (read); no meaning (write) 1 An explicit sense cycle has finished (read); clear bit (write)

25.4.2 IIM Status Interrupt Mask Register (IIM_SIMR)

IIM_SIMR masks program and sense cycle done interrupts.

Address: 0xFC0C_8004 (IIM_SIMR) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	PRGDM	SNSDM
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-3. IIM Status Interrupt Mask Register (IIM_SIMR)

Table 25-3. IIM_SIMR Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1 PRGDM	Program mask. Masks interrupt generation due to PRGD events. 0 PRGD events do not cause an interrupt 1 PRGD events cause an interrupt
0 SNSDM	Explicitly sense cycle done mask. Masks interrupt generation due to SNSD events. 0 SNSD events do not cause an interrupt 1 SNSD events cause an interrupt

25.4.3 IIM Error Status Register (IIM_ESR)

IIM_ESR reports any error conditions when using the IIM.

Address: 0xFC0C_8008 (IIM_ESR) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	WPE	OPE	RPE	0	SNSE	PARE	
W								
Reset	0	0	0	0	0	0	—	—

Figure 25-4. IIM Error Status Register (IIM_ESR)

Table 25-4. IIM_ESR Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 WPE	Write protect error. Indicates an e-fuse program operation was attempted to a write-protected fuse bank, or a locked byte, or when the value of IIM_PRGP is not 0XAA. When this bit is set an interrupt is generated if IIM_EIMR[WPEM] is set. This bit is automatically set by hardware and must be cleared by writing 1 to it. 0 There was no write-protect error (read); no meaning (write) 1 There was a write-protect error (read); clear bit (write)
5 OPE	Override protect error. Indicates an attempt was made to override the values in an override-protected fuse bank, or a locked byte. When this bit is set an interrupt is generated if IIM_EIMR[OPEM] is set. This bit is automatically set by hardware and must be cleared by writing 1 to it. 0 There was no override-protect error (read); no meaning (write) 1 There was an override-protect error (read); clear bit (write)
4 RPE	Read protect error. Indicates an attempt was made to read values from a read-protected fuse bank or SCC. When this bit is set an interrupt is generated if IIM_EIMR[RPEM] is set. This bit is automatically set by hardware and must be cleared by writing 1 to it. 0 There was no read-protect error (read); no meaning (write) 1 There was a read-protect error (read); clear bit (write)
3	Reserved, must be cleared.
2 SNSE	Explicit sense cycle error. Indicates an explicit fuse sense was refused, because IIM_FBAC*n*[FBESP] is set, or more than one bit of IIM_FCR[SNSN, SNS1, SNS0, PRG] are set at the same time. When this bit is set an interrupt is generated if IIM_EIMR[SNSEM] is set. This bit is automatically set by hardware and must be cleared by writing 1 to it. 0 There was no explicit sense error (read); no meaning (write) 1 There was an explicit sense error (read); clear bit (write)
1 PARE	Parity error of cache. Indicates a parity error was detected in the hardware or software fuse cache. When this bit is set an interrupt is generated if IIM_EIMR[PAREM] is set. This bit is automatically set by hardware and must be cleared by writing 1 to it. 0 There was no explicit sense error (read); no meaning (write) 1 There was an explicit sense error (read); clear bit (write)
0	Reserved, must be cleared.

25.4.4 IIM Error Interrupt Mask Register (IIM_EIMR)

IIM_EIMR masks any of the error interrupts indicated by IIM_ESR.

Address: 0xFC0C_800C (IIM_EIMR) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	WPEM	OPEM	RPEM	0	SNSEM	PAREM	0
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-5. IIM Error Interrupt Mask Register (IIM_EIMR)

Table 25-5. IIM_EIMR Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 WPEM	Write protect error mask. Masks or unmasks interrupt generation due to WPE events. 0 WPE events do not cause an interrupt 1 WPE events cause an interrupt
5 OPEM	Override protect error mask. Masks or unmasks interrupt generation due to OPE events. 0 OPE events do not cause an interrupt 1 OPE events cause an interrupt
4 RPEM	Read protect error mask. Masks or unmasks interrupt generation due to RPE events. 0 RPE events do not cause an interrupt 1 RPE events cause an interrupt
3	Reserved, must be cleared.
2 SNSEM	Explicit sense cycle error mask. Masks or unmasks interrupt generation due to SNSE events. 0 SNSE events do not cause an interrupt 1 SNSE events cause an interrupt
1 PAREM	Parity error of cache mask. Masks or unmasks interrupt generation due to PARE events. 0 PARE events do not cause an interrupt 1 PARE events cause an interrupt

25.4.5 IIM Fuse Control Register (IIM_FCR)

IIM_FCR controls fuse programming and explicit sense cycles.

Address: 0xFC0C_8010 (IIM_FCR)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	DPC	PRG_LENGTH			0	0	0	0
W					ESNSN	ESNS0	ESNS1	PRG
Reset	0	0	1	1	0	0	0	0

Figure 25-6. IIM Fuse Control Register (IIM_FCR)

Table 25-6. IIM_FCR Field Descriptions

Field	Description
7 DPC	Delayed program cycle. Selects immediate or delayed program. When this bit is cleared, program cycles start immediately upon setting of PRG. When this bit is set, program cycles are delayed until MISCCR2[DPS] is set. See Chapter 9, “Chip Configuration Module (CCM)”, for details. 0 Program cycles begin immediately upon setting PRG 1 Program cycles are delayed; they do not begin until MISCCR2[DPS] is set
6–4 PRG_LENGTH	Program length. Defines the length of program pulse: PRG_LENGTH × (peroid of 32 kHz clock)

Table 25-6. IIM_FCR Field Descriptions (continued)

Field	Description
3 ESNS_N	Explicit sense—normal. Setting this bit initiates an unstressed (normal) explicit sense cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when sense operation completes. Only one of ESNS_N, ESNS_0, ESNS_1, PRG can be set, otherwise IIM_ESR[SNSE] is set to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate an unstressed explicit sense cycle (write)
2 ESNS_0	Explicit Sense—0 Stressed. Setting this bit initiates a 0-stressed explicit sense cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when sense operation completes. Only one of ESNSN, ESNS0, ESNS1, or PRG can be set, else IIM_ESR[SNSE] is set to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate a 0-stressed explicit sense cycle (write)
1 ESNS_1	Explicit sense—1 stressed. Setting this bit initiates a 1-stressed explicit sense cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when sense operation completes. Only one of ESNSN, ESNS0, ESNS1, PRG can be set, otherwise IIM_ESR[SNSE] is set to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate a 1-stressed explicit sense cycle (write)
0 PRG	Program. Setting this bit initiates a fuse program cycle. Reading of this bit always returns zero. FSM generates a done signal when the operation completes. This bit is cleared automatically by hardware when program operation completes. Only one of ESNSN, ESNS0, ESNS1, PRG can be set, otherwise IIM_ESR[SNSE] is set to indicate this error. 0 Return 0 for all read (read); No meaning (write) 1 Initiate a program cycle (write)

25.4.6 IIM Upper Address Register (IIM_UA)

This register contains the top part of the address of the e-fuse bit to be programmed or the byte to be sensed in an explicit sense cycle. Programming is done on a bit basis, so the program address is a full bit address. Sensing is done on a byte basis, so the bottom three bits of the address are ignored.

Address: 0xFC0C_8014 (IIM_UA) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	0	0	A[13:8]					
W								
Reset	0	0	0	0	0	0	0	0

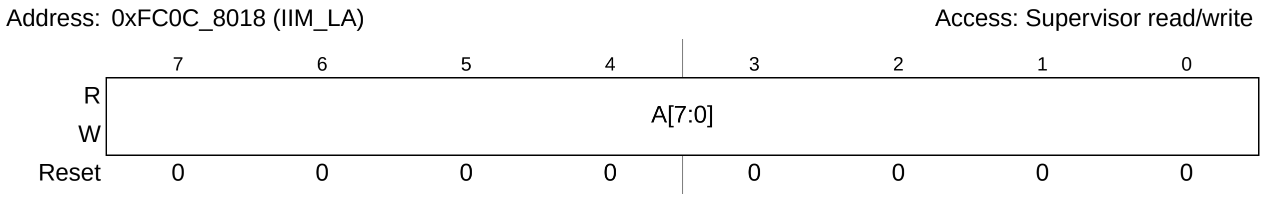
Figure 25-7. Upper Address Register (IIM_UA)

Table 25-7. IIM_UA Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 A[13:8]	The top six bits of the address of the e-fuse bit to be programmed or the word to be sensed explicitly. The address must be written prior to setting IIM_FCR[PRG or ESNS*x*] to initiate the program/sense operation. A[13:11] selects the fuse bank. A[10:8] provide the most significant portion of the row address within the bank.

25.4.7 IIM Lower Address Register (IIM_LA)

This register contains the bottom 8 bits of the address of the e-Fuse bit to be programmed or word to be explicitly sensed.



Address: 0xFC0C_8018 (IIM_LA) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R W	A[7:0]							
Reset	0	0	0	0	0	0	0	0

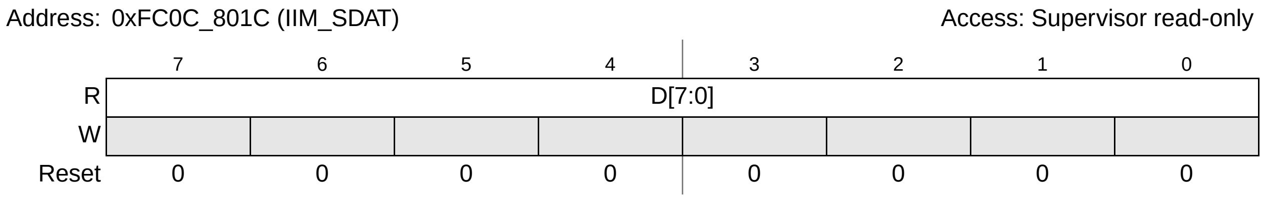
Figure 25-8. IIM Lower Address Register (IIM_LA)

Table 25-8. IIM_LA Field Descriptions

Field	Description
7–0 A[7:0]	The bottom eight bits of the address of the e-fuse bit to be programmed or word to be sensed explicitly. The address must be written prior to setting IIM_FCR[PRG or ESNS*x*] to initiate a program or sense operation. A[7:3] provides the least significant portion of the row address. A[2:0] selects the bit position within the selected row.

25.4.8 Explicit Sense Data Register (IIM_SDAT)

On an explicit sense cycle, the data sensed from the fuses is placed in this register at the conclusion of the sense cycle. Software can recognize the conclusion of the sense cycle by IIM_SR[SNSD].



Address: 0xFC0C_801C (IIM_SDAT) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R	D[7:0]							
W								
Reset	0	0	0	0	0	0	0	0

Figure 25-9. IIM Explicit Sense Data Register (IIM_SDAT)

Table 25-9. IIM_SDAT Field Descriptions

Field	Description
7–0 D[7:0]	The data sensed from the fuses. Setting is unknown.

25.4.9 Program Protection Register (IIM_PRGP)

This register prevents accidental fuse programming. The fuses can be blown only when the value of this register is 0xAA. Software should only program this register to 0xAA while actively blowing fuses. After the program operation is complete, this register should be immediately reprogrammed to a different value.

Address: 0xFC0C_8028 (IIM_PRGP) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R W	PRGP							
Reset	0	0	0	0	0	0	0	0

Figure 25-10. IIM Program Protection Register (IIM_PRGP)

Table 25-10. IIM_PRGP Field Descriptions

Field	Description
7–0 PRGP	The fuses are blown only when the value of this register is 0xAA. Any attempt to program the fuse while the value is other than 0xAA is terminated with error, IIM_ESR[WPE] is set.

25.4.10 IIM Divide Factor Register (IIM_DIVIDE)

IIM_DIVIDE provides a divide factor to generate a 1 MHz clock from peripheral clock. This clock is further divided in the IIM to generate a 32 kHz clock for the fusebox. The value of this register is the frequency of the peripheral clock; however, the minimum is 1.

Address: 0xFC0C_803C (IIM_DIVIDE) Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R W	DIVIDE							
Reset	0	1	0	0	0	0	1	0

Figure 25-11. IIM Divide Factor Register (IIM_DIVIDE)

Table 25-11. IIM_DIVIDE Field Descriptions

Field	Description
7–0 DIVIDE	This register provides a divide factor with which the peripheral clock is divided to a 1 MHz clock. The 1 MHz clock is further divided to 32 kHz. The default vaule is 66 (0x42).

25.4.11 Fuse Banks 0–7 Registers

Each fuse bank consists of a protection register (IIM_FBAC*n*) and user-definable areas. Fuse banks 0–1 are read-only and banks 2–7 are software programmable.

- Reading each of these bits returns the fuse state (0 = unblown; 1 = blown), if the corresponding FBAC*n*[FBRP] is cleared (unblown). Disallowed reads always return zero and cause IIM_ESR[RPE] to set.

- Writing these bits overrides the values without modifying the fuse elements if the corresponding FBAC*n*[FBOP] is cleared (unblown). Disallowed override attempts are ignored and cause IIM_ESR[OPE] to set.
- The fuse elements may be programmed (blown) using the fuse programming sequence, if the corresponding FBAC*n*[FBWP] is cleared (unblown). Disallowed programming attempts are ignored and cause IIM_ESR[WPE] to set.

25.4.11.1 Fuse Bank *n* Protection Registers (IIM_FBAC*n*)

These registers hold the fuse bank access protection information and correspond to the first word in each fuse bank.

Address: 0xFC0C_8800 (IIM_FBAC0)
0xFC0C_8C00 (IIM_FBAC1)
0xFC0C_9000 (IIM_FBAC2)
0xFC0C_9400 (IIM_FBAC3)
0xFC0C_9800 (IIM_FBAC4)
0xFC0C_9C00 (IIM_FBAC5)
0xFC0C_A000 (IIM_FBAC6)
0xFC0C_A400 (IIM_FBAC7)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	FBWP	FBOP	FBRP		FBESP			
W								
Reset	—	—	—	—	—	—	—	—

Figure 25-12. IIM Fuse Bank *n* Protection Registers (IIM FBAC*n*)

Table 25-12. IIM_FBAC*n* Field Descriptions

Field	Description
7 FBWP	Fuse bank write protect. Controls whether this fuse bank may be programmed. 0 (Unblown) Fuse bank may be programmed 1 (Blown) Fuse bank is write-protected
6 FBOP	Fuse bank override protect. Controls whether this fuse bank may be overridden. 0 (Unblown) Fuse bank may be overridden 1 (Blown) Fuse bank is override-protected
5 FBRP	Fuse bank read protect. Controls whether this fuse bank may be read. 0 (Unblown) Fuse bank may be read by software 1 (Blown) Fuse bank is read-protected
4	Reserved, must be cleared.
3 FBESP	Fuse banks explicit sense protect. Controls whether this fuse bank may be explicitly sensed. The state of this fuse controls whether the IIM state machine allows explicit sense cycles (normal, 0-stress, or 1-stress). 0 (Unblown) Fuse bank may be explicitly sensed by software 1 (Blown) Fuse bank is sense-protected
2–0	Reserved, must be cleared.

25.4.11.2 Codec Analog Test Select Register (IIM_ANATEST)

This register corresponds to the second word in fuse bank 0, which holds the codec analog test select value. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8804 (IIM_ANATEST) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R					ANATESTSEL			
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-13. Codec Analog Test Select Register (IIM_ANATEST)

Table 25-13. IIM_ANATEST Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3–0 ANATESTSEL	Codec analog test select.

25.4.11.3 Codec Bandgap Trim Register (IIM_BGPTRIM)

This register corresponds to the third word in fuse bank 0, which holds the PMU/CODEC bandgap trim value. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8808 (IIM_BGPTRIM) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R	PMUTRIM				BGPTRIM			
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-14. Codec Bandgap Trim Register (IIM_BGPTRIM)

Table 25-14. IIM_BGPTRIM Field Descriptions

Field	Description
7–4 PMUTRIM	PMU trimming select.
3–0 BGPTRIM	Codec bandgap trimming select.

25.4.11.4 Amplifier Bypass Register (IIM_AMPBR)

This register corresponds to the fourth word in fuse bank 0, which holds the bypass select for the codec amplifiers (microphone, speaker, handset, headphone). The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_880C (IIM_AMPBR) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R					MIC	HP	HS	SPK
W								
Reset	—	—	—	—	0	0	0	0

Figure 25-15. Amplifiers Bypass Register (IIM_AMPBR)

Table 25-15. IIM_AMPBR Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3 MIC	Micrphone amplifier bypass. 0 Bypass determined by AMPS_BYP register within the codec module. 1 Microphone amplifier is bypassed. The ADC is directly driven by ADC_P and ADC_N.
2 HP	Headphone amplifier bypass. 0 Bypass determined by AMPS_BYP register within the codec module. 1 Headphone amplifier is bypassed. HP_OUT and HP_DUMMY are tri-stated.
1 HS	Handset amplifier bypass. 0 Bypass determined by AMPS_BYP register within the codec module. 1 Handset amplifier is bypassed. The DAC outputs are available on DAC_P and DAC_N.
0 SPK	Speaker amplifier bypass. 0 Bypass determined by AMPS_BYP register within the codec module. 1 Speaker amplifier is bypassed. SPK_OUTN and SPK_OUTP are tri-stated.

25.4.11.5 Driver Ampilfier Control Register (IIM_DRVACR)

This register is the fifth word in fuse bank 0, which holds the driver amplifier controls (thermal limit and alternate common mode). The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8810 (IIM_DRVACR) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R						ACM	THMLMT	
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-16. Driver Amplifier Control Register (IIM_DRVACR)

Table 25-16. IIM_DRVACR Field Descriptions

Field	Description
7–3	Reserved, must be cleared.
2 ACM	Alternate common mode for driver amplifiers. Sets the common mode (CM) for the driver amps to 1.5 V or 1.325 V. The CM for the microphone amp is always 1.325 V. 0 1.325V 1 1.5 V **Note:** Common mode for the microphone amplifier is always 1.325V.
1–0 THMLMT	Thermal limit. Controls the temperature trip point. 00 110°C 01 125°C 10 150°C 11 Bypass thermal detect

25.4.11.6 Speaker Ampilfier Control Register (IIM_SPKACR)

This register corresponds to the sixth word in fuse bank 0, which holds the speaker amplifier controls. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8814 (IIM_SPKACR) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R						HBC	OCL	HFOCL
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-17. Speaker Amplifier Control Register (IIM_SPKACR)

Table 25-17. IIM_SPKACR Field Descriptions

Field	Description
7–3	Reserved, must be cleared.
2 HBC	Speaker high bias current. Controls the bias current in the output stage of the speaker amplifier. 0 250 A (typical) 1 500 A (typical)
1 OCL	Speaker overcurrent limit. 0 700 mA (typical) 1 640 mA (typical)
0 HFOCL	Speaker current half limit. Halves the current limit for the speaker amplifier as specified by OCL. 0 Overcurrent limit is determined by OCL 1 Overcurrent limit is half of what is set by OCL.

25.4.11.7 Handset and Headphone Ampilfier Control Register (IIM_HSHPACR)

This register is the seventh word in fuse bank 0, which holds the handset and headphone amplifier controls. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8818 (IIM_HSHPACR) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R					HS HBC	HS OCL	HP HBC	HP OCL
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-18. Handset and Headphone Amps Register (IIM_HSHPACR)

Table 25-18. IIM_HSHPACR Field Descriptions

Field	Description
7–4	Reserved, must be cleared.
3 HSHBC	Handset high bias current. Controls the bias current in handset amplifier output stage. 0 250 A (typical) 1 500 A (typical)
2 HSOCL	Handset overcurrent limit. Controls the overcurrent limit of the handset amplifier. 0 375 mA (typical) 1 325 mA (typical)
1 HPHBC	Headphone high bias current. Controls the bias current in headphone amplifier output stage. 0 250 A (typical) 1 500 A (typical)
0 HPOCL	Headphone overcurrent limit. Controls the overcurrent limit of the headphone amplifier. 0 175 A (typical) 1 125 A (typical)

25.4.11.8 Security Key Registers (IIM_SECKEY*n*)

IIM_SECKEY*n* contain the security controller’s 168-bit secure key split across 21 registers. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_881C (IIM_SECKEY0) Access: Supervisor read-only
0xFC0C_8820 (IIM_SECKEY1)
...
0xFC0C_8868 (IIM_SECKEY19)
0xFC0C_886C (IIM_SECKEY20)

	7	6	5	4	3	2	1	0
R	KEY							
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-19. Secure Key Registers (IIM_SECKEY*n*)

25.4.11.9 Self Adjust for KRAM Register (IIM_SAKRAM)

This register generates the self adjustment bits for KRAM memory. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8874 (IIM_SAKRAM) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R			SELFADJ					
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-20. Self Adjust for KRAM Register (IIM_SAKRAM)

Table 25-19. IIM_SAKRAM Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 SELFADJ	Self adjust bits for KRAM memory.

25.4.11.10 Self Adjust for TAG Register (IIM_SATAG)

This register generates the self adjustment bits for TAG memory. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8878 (IIM_SATAG) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R			SELFADJ					
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-21. Self Adjust for TAG Register (IIM_SATAG)

Table 25-20. IIM_SATAG Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 SELFADJ	Self adjust bits for TAG memory.

25.4.11.11 Self Adjust for Cache Register (IIM_SACACHE)

This register generates the self adjustment bits for cache memory. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_887C (IIM_SACACHE) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R			SELFADJ					
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-22. Self Adjust for Cache Register (IIM_SACACHE)

Table 25-21. IIM_SACACHE Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 SELFADJ	Self adjust bits for cache memory.

25.4.11.12 Self Adjust for SCCM Register (IIM_SASCCM)

This register generates the self adjustment bits for SCCM memory. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8C04 (IIM_SASCCM) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R			SELFADJ					
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-23. Self Adjust for SCCM Register (IIM_SASCCM)

Table 25-22. IIM_SASCCM Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 SELFADJ	Self adjust bits for SCCM memory.

25.4.11.13 Chip Device ID Select Register (IIM_DEVID)

This register selects the device ID based on specifc values of these fuses. The fuses associated with this register are sensed-out when the IIM comes out of reset.

Address: 0xFC0C_8C08 (IIM_DEVID) Access: Supervisor read-only

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

Figure 25-24. Device ID Select Register (IIM_DEVID)

Table 25-23. IIM_DEVID Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1–0 ID	Device ID select. Selects the device ID based on specific value of the fuses.

25.4.11.14 Fuse Bank *n* Data Register (FB*n*W*m*)

Fuse bank 1 contains 29 bytes of unused read-only hardware fuses.

Address: 0xFC0C_8C0C—0xFC0C_8C7C (IIM_FB1W3–31) Access: Supervisor read-only

	7	6	5	4	3	2	1	0
R	DATA							
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-25. Fuse Bank 1 Data Register (FB1W3–FB1W31)

Fuse banks 2–7 contain 31 bytes of software-programmable fuses for general-purpose use.

Address: 0xFC0C_9004—0xFC0C_907C (IIM_FB2W1–31)
0xFC0C_9404—0xFC0C_947C (IIM_FB3W1–31)
0xFC0C_9804—0xFC0C_987C (IIM_FB4W1–31)
0xFC0C_9C04—0xFC0C_9C7C (IIM_FB5W1–31)
0xFC0C_A004—0xFC0C_A07C (IIM_FB6W1–31)
0xFC0C_A404—0xFC0C_A47C (IIM_FB7W1–31)

Access: Supervisor read/write

	7	6	5	4	3	2	1	0
R	DATA							
W								
Reset	–	–	–	–	–	–	–	–

Figure 25-26. Fuse Bank 2–7 Data Register (FB2W1–FB7W31)

25.5 Functional Description

The IIM is an 8-bit peripheral which implements interfaces for fuses, as well as the hardware revision codes, cryptographic keys, and miscellaneous system-level feature-enabling circuitry. Up to eight banks, each with up to 256 fuses, are supported.

25.5.1 Fuse Bank 0–1

Fuse bank 0 (addresses 0xFC0C_8804–FC0C_887C) is reserved for codec, amplifiers, security key storage, etc.

The first two bytes of fuse bank 1 (addresses 0xFC0C_8C04–FC0C_8C08) are reserved for SCCM memory and the device ID. The remaining area in fuse bank 1 is reserved for future hardware-visible registers.

25.5.2 Fuse Bank 2–7

Fuse bank 2–7 are available for user-specifc fuses.

25.5.3 Fuse Value Storage

The values of fuses may be read from one of three places:

- Hardware-visible fuse value shadow cache
- Software fuse value shadow cache
- Fuse elements themselves

The fuse values are cached to reduce the risk of accidental programming of fuses due to repeated reads, and to reduce power consumption associated with sense cycles.

25.5.3.1 Hardware-Visible Fuse Shadow Cache

The hardware-visible fuse includes fuses in bank 0–1, and FBAC*n* words of each bank. The IIM senses these fuses and writes their values to the appropriate registers when it comes out of reset, and ensures that any change to the fuses is also immediately reflected in the registers. The overall HW-visible word read sequence is shown in Section 25.5.5.1, “Read Sequence”. Each word has a parity bit as shown in Figure 25-27.

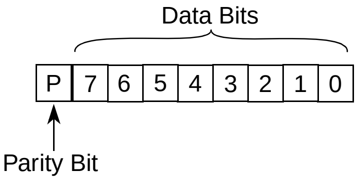


Figure 25-27. Hardware Cache Word Format

If a parity error is detected on the shadow cache, the IIM sets IIM_ESR[PARE] and re-initializes cache from the fuses.

25.5.3.2 Software Fuse Value Shadow Cache

The software fuses include fuses in bank 2–7, and FBAC*n* words of each bank. Each word in the cache RAM includes a valid bit and a parity bit, as shown in Figure 25-28. The valid bit sets if the data and parity bits are set according to the sensed state of the corresponding fuses. Odd parity is used across the data bits only (the valid bit is not included in the parity calculation).

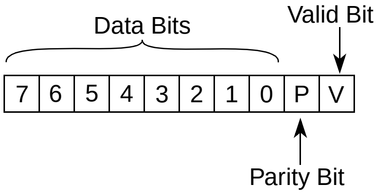


Figure 25-28. Software Cache Word Format

All bits are cleared when the IIM comes out of reset. When a fuse word is read, the IIM first attempts to read it from the cache (except for the hardware-visible fuses, see Section 25.5.3.1, “Hardware-Visible Fuse Shadow Cache”). A sense cycle is only run to the fuses if the cached word is invalid or the parity is incorrect. The overall cached fuse word read sequence is shown in Section 25.5.5.1, “Read Sequence”. The cache words and fuse words are one-to-one mapped.

25.5.4 Fuse Protection

Each fuse bank contains a set of protection bits located in IIM_FBAC*n*. These bits are defined as:

- FBWP—controls whether the bank can be programed
- FBOP—controls whether the bank can be overridden
- FBRP—controls whether the bank can be read
- FBESP—controls whether the bank can be explicitly sensed

25.5.5 Fuse Bank Operations

25.5.5.1 Read Sequence

Fuses may be read from any bank that is not read-inhibited (IIM_FBAC*n*[FBRP] is unblown). The read sequence from a hardware-visible signals word is shown below.

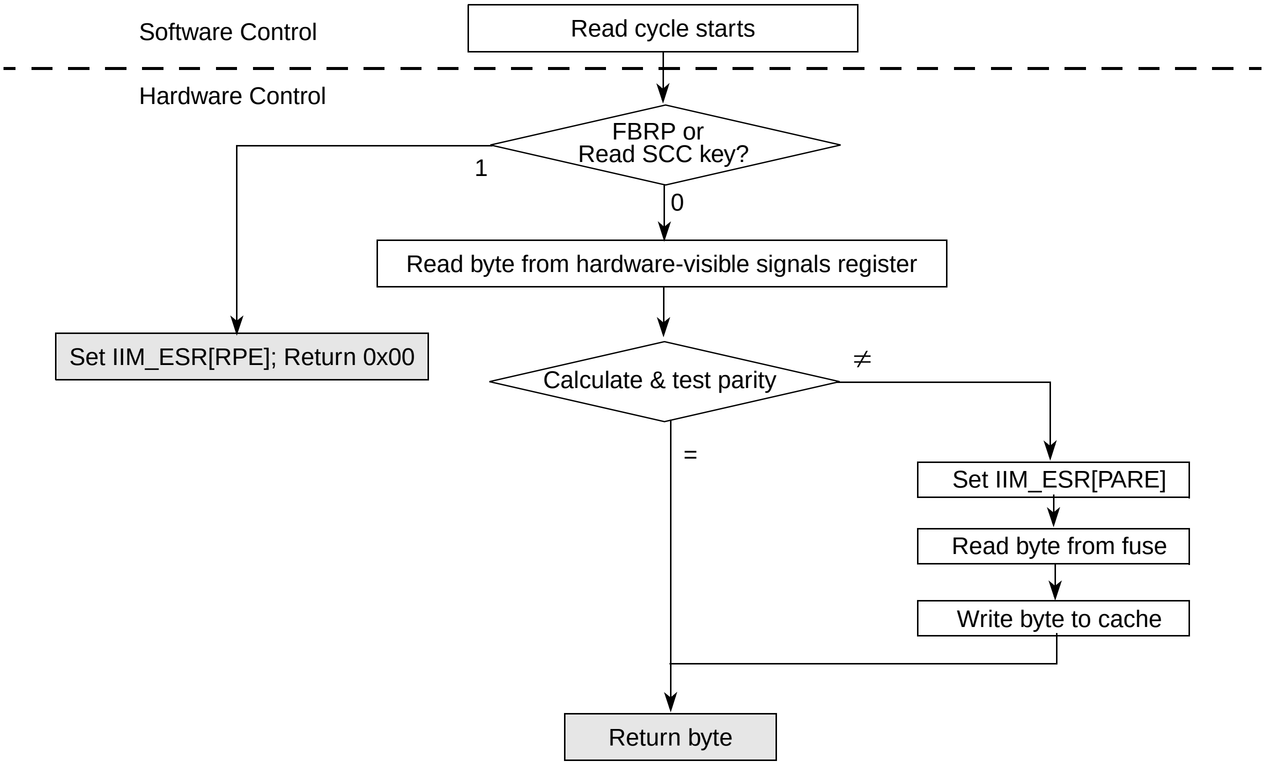


Figure 25-29. Hardware-Visible Fuse Read

The read sequence of a cacheable fuse word is shown below.

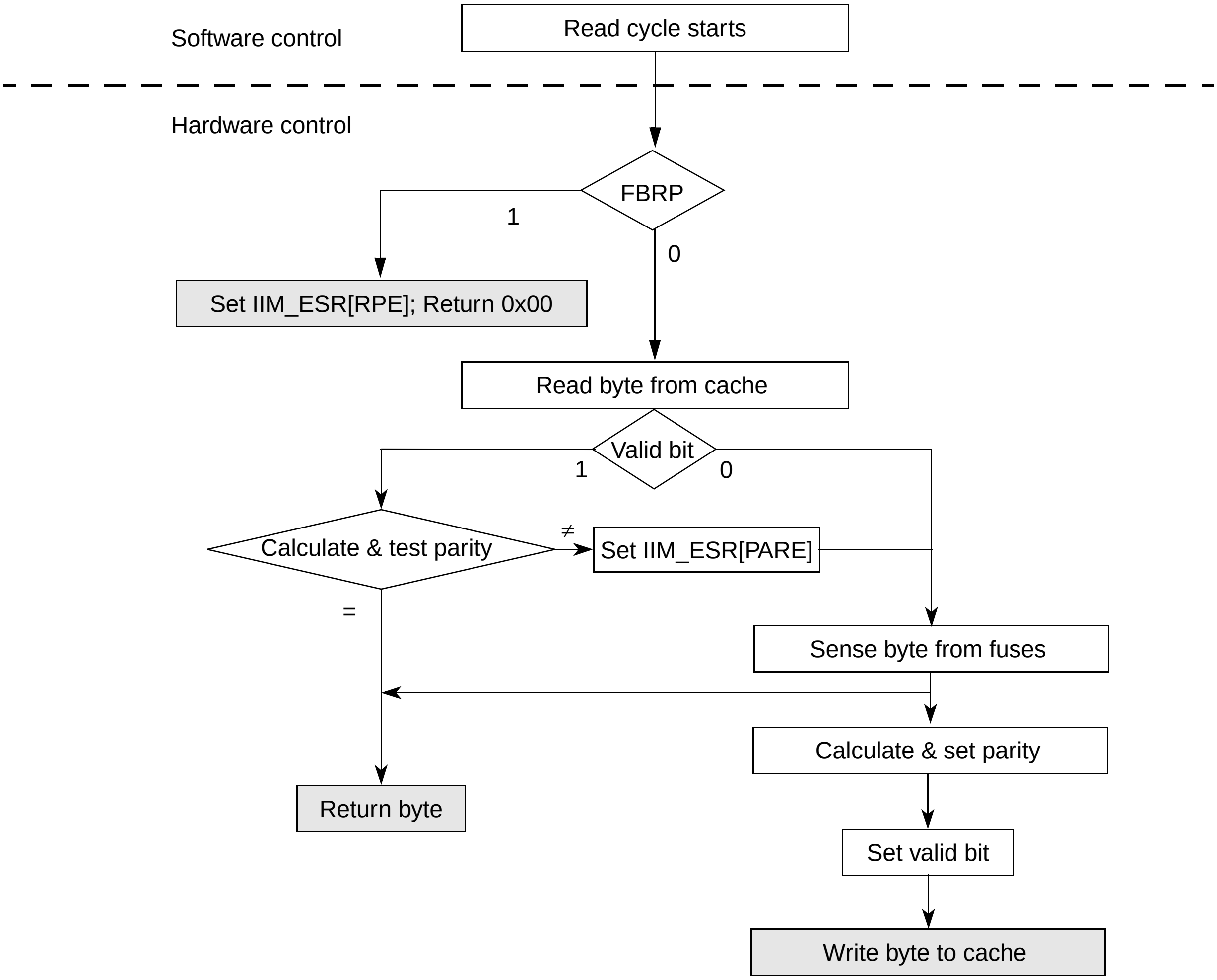


Figure 25-30. Software Fuse Read

25.5.5.2 Explicit Sense Sequence

The explicit sense sequence is depicted below.

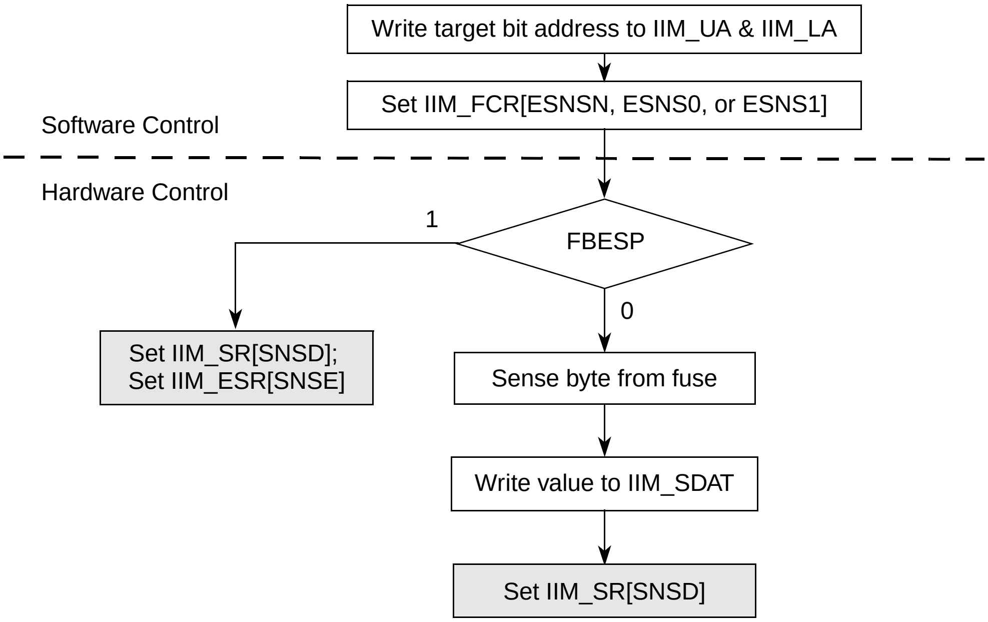


Figure 25-31. Explicit Sense Sequence

25.5.5.3 Programming Sequence

The software-controlled e-fuse programming sequence is depicted in Figure 25-32.

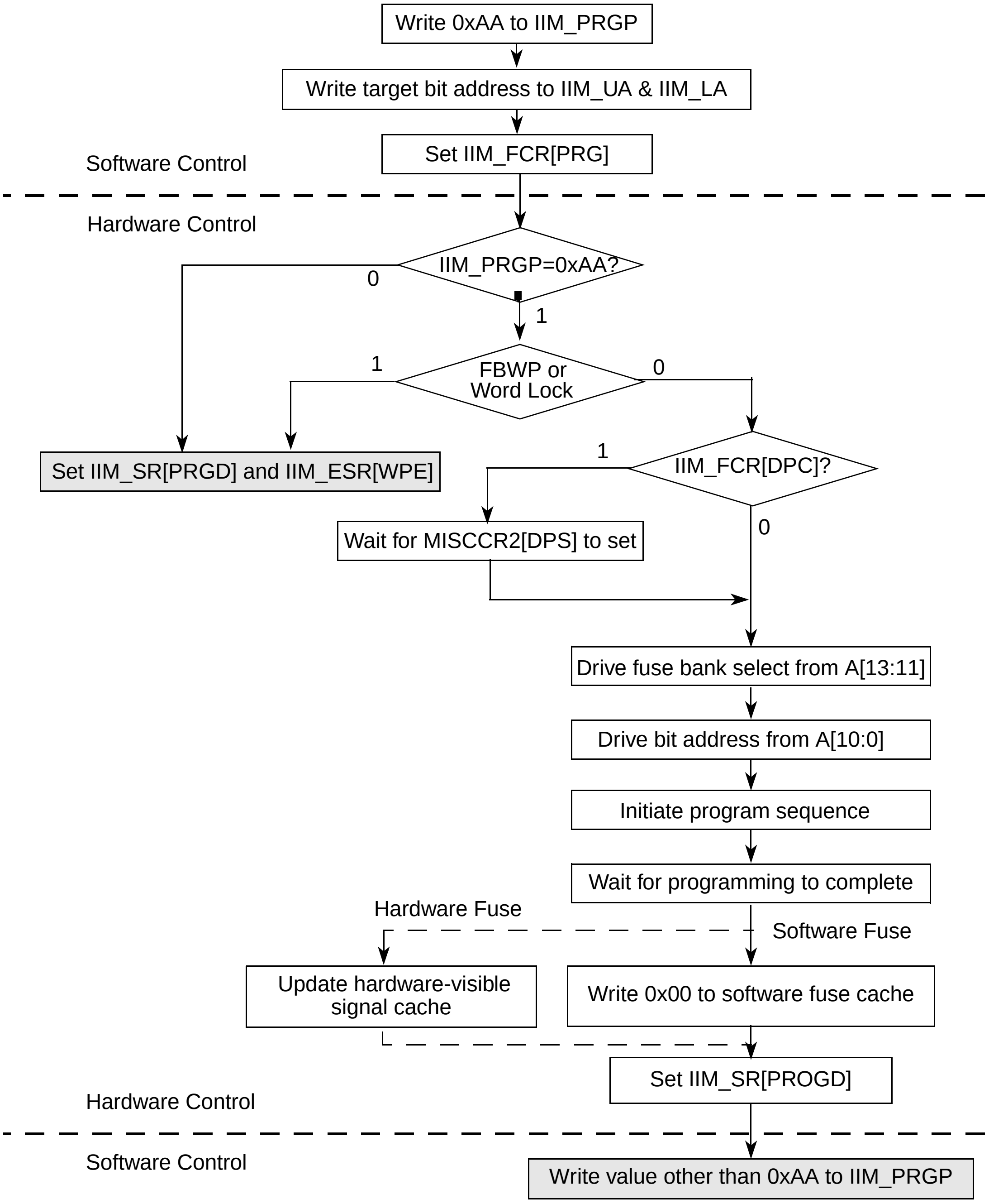

Write 0xAA to IIM_PRGP
Write target bit address to IIM_UA & IIM_LA
Set IIM_FCR[PRG]
Software Control
Hardware Control
IIM_PRGP=0xAA?
0
1
FBWP or Word Lock
1
0
Set IIM_SR[PRGD] and IIM_ESR[WPE]
IIM_FCR[DPC]?
1
0
Wait for MISCCR2[DPS] to set
Drive fuse bank select from A[13:11]
Drive bit address from A[10:0]
Initiate program sequence
Wait for programming to complete
Hardware Fuse
Software Fuse
Update hardware-visible signal cache
Write 0x00 to software fuse cache
Set IIM_SR[PROGD]
Hardware Control
Software Control
Write value other than 0xAA to IIM_PRGP

Figure 25-32. e-Fuse Program Sequence

25.5.5.4 Override Sequence

The override sequence is depicted in Figure 25-33.

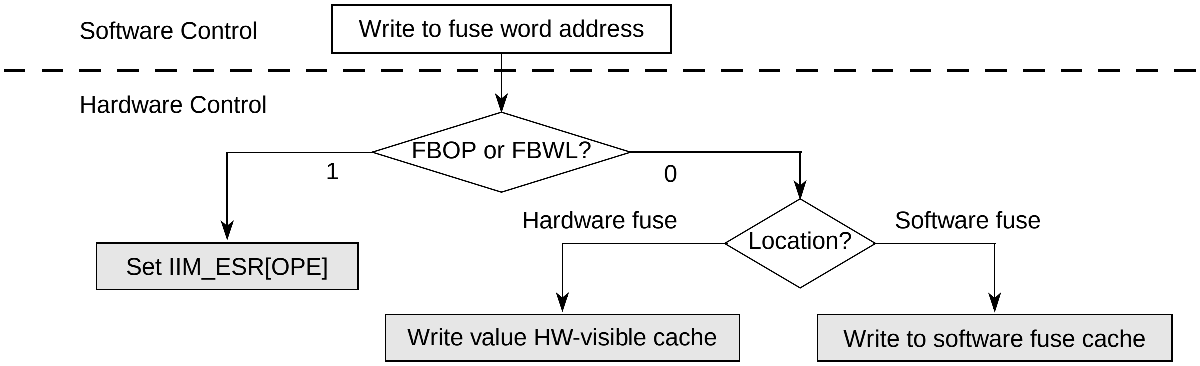


Figure 25-33. Fuse Value Override Sequence

25.6 Initialization/Application Information

When the IIM comes out of reset, IIM automatically senses the hardware-visible fuses, and writes their values into the appropriate registers. The IIM also senses IIM_FBAC*n* from each of the fuse banks. Software fuses are not sensed until software requests a read for them. The software cache is cleared on reset.

25.6.1 Program

A wait period (>20 µs) is required between program and read operations.

Chapter 26
Subscriber Identification Module (SIM)

26.1 Introduction

The subscriber identification module (SIM) facilitates communication to SIM cards or Eurochip pre-paid phone cards. The SIM module has two ports that can be used to interface with the various cards.

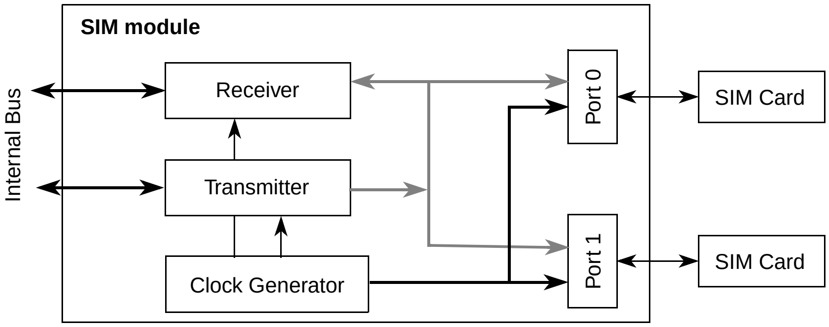


Figure 26-1. SIM Block Diagram

26.2 Overview

The SIM design is summarized in the following sections.

26.2.1 Features

The SIM contains the following features:

- Programmable clock divisor for SIM card clock generation
- Transmitter block with a transmit state machine, transmit shift register, and transmit FIFO
  - 16-byte transmit FIFO
  - Automatic NACK generation on parity and overrun errors
  - Hardware data format support (inverse convention or direct convention)
  - Retransmission of data upon SIM card NACK request with programmable maximum threshold of retransmissions
  - Programmable guard time between transmitted bytes
- Receiver block with a receive state machine, receive FIFO, and control logic
  - 32-byte receive FIFO
  - Decoding of initial character mode for setting data format
  - Hardware data format support (inverse convention or direct convention)

  - — NACK detection
  - — 11 ETU character support
  - — Character wait time counter
- Supports numerous port control functions necessary for SIM card interaction
  - — SIM card presence detect with interrupt capability
  - — Deep sleep wake-up via SIM card presence detect interrupt
  - — Manual control of all SIM card interface signals
  - — Automatic power-down of port logic on SIM card presence detect

26.2.2 Modes of Operation

The SIM module I/O interface operates in one mode summarized below.

- Internal one wire interface. In this mode the TX pin is routed to the smart card. The RX signal is connected to the TX pin internal to the processor. The 3VOLT bit in the port control register is set and SIM_ODCR[ODP*n*] is set. For this interface to work properly the TX pin must be pulled high by a resistor. The value should be selected small enough to give a fast rise time.

26.3 External Signal Description

See Table 26-1 for a summary of the SIM module signals.

Table 26-1. SIM Signal Descriptions

Signal	I/O	Description
SIM_CLK[1:0]	O	Clock for the smart card. Typical frequencies are 1–5 MHz. This clock is 372 times the data rate that is on SIM_DATA. There is no required timing relationship between this clock signal and any of the other data signals. This is because of the asynchronous nature of the protocol.
SIM_RST[1:0]	O	Reset signal.
SIM_VEN[1:0]	O	Power supply enable signal.
SIM_DATA[1:0]	I/O	Bidirectional transmit/receive data signal.
SIM_PD[1:0]	I	Card insertion detect signal.

26.4 Memory Map/Register Definition

Table 26-2 shows the SIM memory map.

Table 26-2. SIM Memory Map

Address	Register	Access	Reset Value	Section/Page
0xFC0A_C000	SIM port 1 control register (SIM_CR1)	R/W	0x0000_0000	26.4.1/26-4
0xFC0A_C004	SIM setup register (SIM_SETUP)	R/W	0x0000_0000	26.4.2/26-5
0xFC0A_C008	SIM port 1 detect register (SIM_DETECT1)	R/W	0x0000_----	26.4.3/26-6
0xFC0A_C00C	SIM port 1 transmit buffer register (SIM_TBUF1)	R/W	0x0000_0000	26.4.4/26-7

Table 26-2. SIM Memory Map (continued)

Address	Register	Access	Reset Value	Section/Page
0xFC0A_C010	SIM port 1 receive buffer register (SIM_RBUF1)	R	0x0000_0000	26.4.5/26-7
0xFC0A_C014	SIM port 0 control register (SIM_CR0)	R/W	0x0000_0000	26.4.1/26-4
0xFC0A_C018	SIM control register (SIM_CR)	R/W	0x0000_0006	26.4.6/26-8
0xFC0A_C01C	SIM clock prescaler register (SIM_PRE)	R/W	0x0000_0002	26.4.7/26-10
0xFC0A_C020	SIM receive threshold register (SIM_RTHR)	R/W	0x0000_0001	26.4.8/26-11
0xFC0A_C024	SIM enable register (SIM_EN)	R/W	0x0000_0000	26.4.9/26-11
0xFC0A_C028	SIM transmit status register (SIM_TSR)	R/W	0x0000_00B8	26.4.10/26-12
0xFC0A_C02C	SIM receive status register (SIM_RSR)	R/W	0x0000_0040	26.4.11/26-13
0xFC0A_C030	SIM interrupt mask register (SIM_IMR)	R/W	0x0000_1FFF	26.4.12/26-15
0xFC0A_C034	SIM port0 transmit buffer register (SIM_TBUF0)	R/W	0x0000_0000	26.4.4/26-7
0xFC0A_C038	SIM port0 receive buffer register (SIM_RBUF0)	R	0x0000_0000	26.4.4/26-7
0xFC0A_C03C	SIM port0 detect register (SIM_DETECT0)	R/W	0x0000_----	26.4.3/26-6
0xFC0A_C040	SIM data format register (SIM_FORMAT0)	R/W	0x0000_0000	26.4.13/26-17
0xFC0A_C044	SIM transmit threshold register (SIM_TTHR)	R/W	0x0000_0000	26.4.14/26-17
0xFC0A_C048	SIM transmit guard control register (SIM_TGCR)	R/W	0x0000_0000	26.4.15/26-18
0xFC0A_C04C	SIM open drain configuration control register (SIM_ODCR)	R/W	0x0000_0000	26.4.16/26-19
0xFC0A_C050	SIM reset control register (SIM_RCR)	R/W	0x0000_0000	26.4.17/26-19
0xFC0A_C054	SIM character wait time register (SIM_CWTR)	R/W	0x0000_FFFF	26.4.18/26-20
0xFC0A_C058	SIM general purpose counter register (SIM_GPCNT)	R/W	0x0000_FFFF	26.4.19/26-21
0xFC0A_C05C	SIM divisor register (SIM_DIV)	R/W	0x0000_00FF	26.4.20/26-21
0xFC0A_C060	SIM block wait time register (SIM_BWT)	R/W	0x0000_FFFF	26.4.21/26-22
0xFC0A_C064	SIM block guard time register (SIM_BGT)	R/W	0x0000_0000	26.4.22/26-22
0xFC0A_C068	SIM block wait time register high (SIM_BWTH)	R/W	0x0000_FFFF	26.4.23/26-23
0xFC0A_C06C	SIM transmit FIFO status register (SIM_TFSR)	R	0x0000_0000	26.4.24/26-23
0xFC0A_C070	SIM receive FIFO counter register (SIM_RFCR)	R	0x0000_0000	26.4.25/26-24
0xFC0A_C074	SIM receive FIFO write pointer register (SIM_RFWP)	R	0x0000_0000	26.4.26/26-24
0xFC0A_C078	SIM receive FIFO read pointer register (SIM_RFRP)	R	0x0000_0000	26.4.27/26-24

26.4.1 SIM Port Control Registers (SIM_CR*n*)

Address: 0xFC0A_C000 (SIM_CR1)
0xFC0A_C014 (SIM_CR0)

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	3VOLT	SCSP	SCEN	SRST	STEN	SVEN	SAPD
W									SFPD							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-2. SIM Port 0–1 Control Registers (SIM_CR*n*)

Table 26-3. SIM_CR*n* Field Descriptions

Field	Description
31–8	Reserved
7 SFPD	Force auto power-down. Starts the automatic power down sequence for the port. This bit will autoclear. Reading this bit returns a zero. 0 No effect 1 Start auto power-down
6 3VOLT	External one-wire interface. Configure the port's transmit pin as bi-directional. This allows the port receive pin to be re-used as general purpose I/O. This operation is restricted to bi-directional data SIM cards only. This bit should not be changed while the receiver or transmitter is enabled. 0 Port1 uses both RCV and XMT pins 1 Port1 XMT pin bidirectional. Port1 RCV PIN unused
5 SCSP	SIM card clock stop polarity. Controls the polarity of the idle SIM clock when the clock is disabled by SCEN. This bit is forced to zero by hardware during the auto power down sequence. This forces the clock be a logic 0 when stopped by auto power down as required by ISO 7816 spec. 0 Clock is logic 0 when stopped by SCEN 1 Clock is logic 1 when stopped by SCEN
4 SCEN	SIM card clock enable. Enables the clock to the SIM card. It is forced low by hardware during the auto power down sequence. 0 SIM card clock disabled 1 SIM card clock enabled
3 SRST	SIM card reset. Controls state of reset line to the SIM card. It is forced low by hardware during the auto power down sequence. SIM card reset signals are active low. 0 SIM card reset inactive 1 SIM card reset active
2 STEN	SIM card transmit enable. Enables the XMT data to the SIM card. It can be forced low by hardware during the auto power down sequence. 0 Transmit Data is forced to zero 1 Transmit Data controlled by SIM module

Table 26-3. SIM_CR*n* Field Descriptions (continued)

Field	Description
1 SVEN	SIM card Vcc enable. Controls the state of the SVEN pin on SIM card port. The SVEN pin controls the SIM card Vcc enable in the power management chip. It is forced low by hardware during the auto power down sequence. 0 SIM card voltage disabled 1 SIM card voltage enabled
0 SAPD	SIM card auto power down. Enables the auto power down function. It will be forced low at the end of the auto power down sequence. 0 Auto power down disabled 1 Auto power down enabled

26.4.2 SIM Port 1 Setup Register (SIM_SETUP)

Address: 0xFC0A_C004 (SIM_SETUP)

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	SPS	AMODE
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-3. SIM Setup Register (SIM_SETUP)

Table 26-4. SIM_SETUP Field Descriptions

Field	Description
31–2	Reserved, must be cleared.
1 SPS	SIM card port select. Controls which port the SIM interface uses. **Note:** The AMODE bit must be cleared when the SPS bit is set. 0 Port 0 enabled 1 Port 1 enabled
0 AMODE	Alternate SIM card mode enable. Enables an alternate SIM module to control this port. **Note:** The SPS bit must be cleared to give the alternate SIM module control. 0 Alternate port disabled 1 Alternate port enabled

26.4.3 SIM Port Detect Registers (SIM_DETECT*n*)

Address: 0xFC0A_C008 (SIM_DETECT1)
0xFC0A_C03C (SIM_DETECT0)

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	SPDS	SPDP	SDI	SDIM
W															w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	—	0	1

Figure 26-4. SIM Port 1 Detect Register (SIM_DETECT*n*)

Table 26-5. SIM_DETECT*n* Field Descriptions

Field	Description
31–4	Reserved
3 SPDS	SIM presence detect select. Controls which edge of the SIM_PD pin is used to detect the presence of the SIM card. 0 Falling edge of SIM_PD input 1 Rising edge of SIM_PD input
2 SPDP	SIM_PD input pin status. Reflects the state of the SIM_PD pin. This bit is not a latched register bit, but instead a synchronized version of the state of the SIM_PD pin itself. 0 SIMPD pin is logic low 1 SIMPD pin is logic high
1 SDI	SIM detect interrupt flag. Indicates an insertion or removal of a SIM card has been detected. This bit may generate an interrupt if SDIM is cleared. Write a one to this bit to clear it. 0 No insertion or removal of SIM card detected 1 Insertion or removal of SIM card detected
0 SDIM	SIM detect interrupt mask. Interrupt mask for the SDI interrupt flag. 0 SDI enabled 1 SDI masked
Summary: SPDS determines which edge transition of the SIM_PD pin is used for SIM card presence detection. Presence detection determines if the card has been inserted or removed. The occurrence of the SIMPD1 edge specified by SPDS will cause the following: SDI to be set; if the SDIM mask is clear, an interrupt on SIMIRQ_N; and if SIM_CR*n*[SAPD] is set, an auto power down sequence to begin. If SIM card insertion is expected, SAPD can be cleared to avoid the auto power down sequence. There is no auto power up sequence. The bit SPDP can be used to determine the current state of the SIM_PD pin.	

26.4.4 SIM Port Transmit Buffer Registers (SIM_TBUF*n*)

Address: 0xFC0A_C00C (SIM_TBUF1)
0xFC0A_C034 (SIM_TBUF0)

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	TXBUF							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. SIM Port Transmit Buffer Registers (SIM_TBUF*n*)

Table 26-6. SIM_TBUF*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–8</td><td>Reserved, must be cleared.</td></tr>
<tr><td>7–0
TXBUF</td><td>Port transmit buffer. Write to the next available location in the transmit buffer. Writes to this register are ignored depending on SIM_SETUP[SPS].
<table>
<tr><th>Transmit Buffer</th><th>SIM_SETUP[SPS]</th><th>Writes Ignored?</th></tr>
<tr><td rowspan="2">SIM_TFBUF0[TXBUF]</td><td>0</td><td>No</td></tr>
<tr><td>1</td><td>Yes</td></tr>
<tr><td rowspan="2">SIM_TFBUF1[TXBUF]</td><td>0</td><td>Yes</td></tr>
<tr><td>1</td><td>No</td></tr>
</table>
</td></tr>
</table>

26.4.5 SIM Port Receive Buffer Registers (SIM_RBUF*n*)

Address: 0xFC0A_C010 (SIM_RBUF1)
0xFC0A_C038 (SIM_RBUF0)

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	CWT	FE	PE	RXBUF							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. SIM Port Receive Buffer Registers (SIM_RBUF*n*)

Table 26-7. SIM_RBUF*n* Field Descriptions

Field	Description
31–11	Reserved
10 CWT	Port CWT flag. Indicates that the current byte was late. It is not necessary to clear the byte since it is overwritten by the next byte received at that location of the FIFO. 0 Byte was on time 1 Byte was late

Table 26-7. SIM_RBUF*n* Field Descriptions (continued)

Field	Description
9 FE	Port frame error flag. Indicates a frame error was detected during reception of the current byte read from the port's receiver. This bit cannot generate an interrupt. It is not necessary to clear the byte since it is overwritten by the next byte received at that location of the FIFO. 0 Byte contains no framing error 1 Byte contains a framing error
8 PE	Port parity error flag. Indicates a parity error was detected during reception of the current byte read from the port's receiver. This bit cannot create an interrupt. It is not necessary to clear the byte since it is overwritten by the next byte received at that location of the FIFO. A parity error can create a NACK pulse. 0 Byte contains no parity error (default) 1 Byte contains a parity error
7–0 RXBUF	Port receive buffer. Read from the next location in the receive buffer. Reads from this register return zero when SPS is cleared.

26.4.6 SIM Control Register (SIM_CR)

Address: 0xFC0A_C018 (SIM_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	BWTEN	XMT_CRC_LRC	CRCEN	LRCEN	CWTEN	GPCNT_CLKSEL		BAUD_SEL			0	SAMPLE12	ONACK	ANACK	ICM	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 26-7. SIM Control Register (SIM_CR)

Table 26-8. SIM_CR Field Descriptions

Field	Description
31–16	Reserved
15 BWTEN	Block wait time enable. Enables the BWT and BGT functions. These functions can then be individually selected using the interrupt mask. 0 Disable BWT and BGT 1 Enable BWT and BGT
14 XMT_CRC_LRC	Transmit CRC or LRC. Specifies whether to transmit the redundancy checking data at the end of a transmission (when the FIFO becomes empty). 0 No redundancy check info transmitted 1 Transmit LRC or CRC info when FIFO empties

Table 26-8. SIM_CR Field Descriptions (continued)

Field	Description
13 CRCEN	CRC Enable. Enables the calculation of the 16-bit CRC value for both receiver and transmitter. The result of the calculation is continuously compared to the expected remainder and reflected in SIM_RSR[CRCOK] register. Clearing this bit resets the current CRC residual value in the SIM hardware. 0 16-bit CRC disabled 1 16-bit CRC enabled
12 LRCEN	LRC Enable. Enables the calculation of the 8-bit LRC value for both receiver and transmitter. The result of the calculation is continuously compared to zero and reflected in SIM_RSR[LRCOK]. Clearing this bit resets the current LRC value in the SIM hardware. 0 8-bit linear redundancy checking disabled 1 8-bit linear redundancy checking enabled
11 CWTEN	Character wait time counter enable. Enables the character wait time counter. Clearing this bit resets the counter to zero. 0 Character wait time counter off 1 Character wait time counter on
10–9 GPCNT_ CLKSEL	General purpose counter clock select. Selects which clock source is used by SIM module general purpose counter. The only way to reset the counter is to clear this field. The counter begins counting as soon as the clock input is selected and the clocks are enabled. These input clocks are enabled through other register bits of the SIM module (KILL_CLOCK, RXEN, and TXEN respectively). 00 Disabled/reset 01 Card clock 10 Receive clock 11 ETU clock (transmit clock)
8–6 BAUD_SEL	SIM baud rate select. Selects the asynchronous baud rate divisor of the clock When set to 111, the divisor is set to the value programmed in the DIVISOR register. This allows for more flexible baud rate determination. 000 31 (372/1 Fi/Di) 001 32 (512/2 Fi/Di) 010 16 (512/4 Fi/Di) 011 8 (512/8 Fi/Di) 100 4 (512/16 Fi/Di) 101 2 (512/32 Fi/Di) 110 1 (512/64 Fi/Di) 111 SIM_DIV
5	Reserved
4 SAMPLE12	Sample12. Set the third stage divider. Sets the corresponding sample rate which is the number of times a bit being received is sampled. 0 Divide by 8 1 Divide by 12
3 ONACK	Overrun NACK Enable. 0 NACK generation on overrun is disabled 1 NACK generation on overrun is enabled
2 ANACK	Automatic NACK Enable. Enables NACK generation for parity errors or invalid initial characters when ICM is set. 0 NACK generation on errors disabled 1 NACK generation on errors enabled

Table 26-8. SIM_CR Field Descriptions (continued)

Field	Description
1 ICM	Initial character mode. Enables initial character mode. This bit is automatically cleared by hardware after a valid initial character is received. This bit is set following reset. 0 Initial character mode disabled 1 Initial character mode enabled
0	Reserved

26.4.7 SIM Clock Prescaler Register (SIM_PRE)

Address: 0xFC0A_C01C (SIM_PRE) 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	PRESCALER							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-8. SIM Clock Prescaler Register (SIM_PRE)

Table 26-9. SIM_PRE Field Descriptions

Field	Description
31–8	Reserved
7–0 PRESCALER	Clock prescaler divisor register. The value written to this register will determine the first stage divider setting. If the ipg_clk is 66Mhz, a typical setting would be 0x0e. This would set the SIM card clock to 66Mhz/14 = 4.7Mhz. The duty cycle of divided clock will be between 45% and 55% according to ISO7816 requirement. For the odd divider factor (2K+1), the duty cycle will be K/(2K+1) and (K+1)/(2k+1). So for 0x03, the duty cycle is 33%-66%. For 0x05, the duty cycle is 40%-60%. For 0x07, the duty cycle is 43%-57%, and for 0x09, the duty cycle is 44%-56%. For all other values, the clock duty cycle can meet the ISO7816 requirement. 0x00~0x02 ipg_clk/ 2 0x03 ipg_clk / 3 0x04 ipg_clk / 4 0x05 ipg_clk / 5 0x06 ipg_clk / 6 ... 0xFD ipg_clk / 253 0xFE ipg_clk / 254 0xFF ipg_clk / 255

26.4.8 SIM Receive Threshold Register (SIM_RTHR)

Address: 0xFC0A_C020 (SIM_RTHR)   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	RTH				RDT				
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. SIM Receive Threshold Register (SIM_RTHR)

Table 26-10. SIM_RTHR Field Descriptions

Field	Description
31–13	Reserved
8–5 RTH	Receive Nack threshold. Specifies the number of consecutive NACK's transmitted by the SIM module, for a given character, before the receive threshold error (RTE) flag is triggered. A value of 0 indicates that RTE is never set. When a valid character is received by the SIM, the internal counter keeping track of the NACK count resets to zero for the subsequent byte being received. If SIM_CR[ANACK] is cleared, RTH has no effect.
4–0 RDT	Receive data threshold. Determines the number of unread bytes that must exist in the FIFO to trigger the receive data register full (RDRF) interrupt flag. If the number of unread bytes in the receive FIFO is greater than or equal to the value in RDT, the SIM_RSR[RDRF] flag is set. A value of zero indicates that there must be 285 unread bytes in the FIFO to trigger RDRF. The RDT value can be altered at any time, and the RDRF flag will be updated accordingly.

26.4.9 SIM Enable Register (SIM_EN)

Address: 0xFC0A_C024 (SIM_EN)   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	TXEN	RXEN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-10. SIM Enable Register (SIM_EN)

Table 26-11. SIM_EN Field Descriptions

Field	Description
31–2	Reserved

Table 26-11. SIM_EN Field Descriptions (continued)

Field	Description
1 TXEN	SIM transmit enable. Enables the SIM transmit state machine. When the SIM is being used to receive data, TXEN should be cleared. This bit also enables the XMT_CLK and RCV_CLK inputs to the general purpose counter. **Note:** Setting this bit (transition from 0 to 1) will reset the CRC and LRC values. 0 SIM transmitter disabled 1 SIM transmitter enabled
0 RXEN	SIM receiver enable. Enables the SIM receive state machine. RXEN must be set whenever the SIM module is in use. The SIM module has an automatic receive mode operation that disables the reception of characters when the transmitter is operational. Once the transmitter has completed sending the last character, the receiver is automatically enabled. This bit also enables the RCV_CLK input to the general purpose counter. 0 SIM receiver disabled 1 SIM receiver enabled

26.4.10 SIM Transmit Status Register (SIM_TSR)

Address: 0xFC0A_C028 (SIM_TSR) 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	GP CNT	TDTF	TFO	TC	ETC	TFE	0	0	XTE
W								w1c	w1c		w1c	w1c	w1c			w1c
Reset	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0

Figure 26-11. SIM Transmit Status Register (SIM_TSR)

Table 26-12. SIM_TSR Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8 GPCNT	General purpose counter flag. Indicates when the general purpose counter has reached the value in the GPCNT register. 0 GPCNT time not reached 1 General purpose counter has reached the GPCNT value
7 TDTF	Transmit data threshold flag. Indicates when the number of bytes in the transmit FIFO is less than or equal to the value programmed in SIM_TTHR[TDT]. 0 Number of bytes in FIFO is greater than TDT 1 Number of bytes in FIFO is less than or equal to TDT

Table 26-12. SIM_TSR Field Descriptions (continued)

Field	Description
6 TFO	Transmit FIFO overfill error. Indicates when the Transmit FIFO has been written with more than 16 bytes. This bit is only cleared by setting either SIM_RCR[FLUSH_XMT, SOFT_RESET]. 0 No transmit FIFO overfill error occurred 1 A Transmit FIFO overfill error occurred
5 TC	Transmit complete. Indicates the SIM transmitter is ready for a new transmission. This bit is set after the guard time has expired for the last byte in the transmit FIFO. TC generates an interrupt if SIM_IMR[TCIM] is cleared. Write one to TC to clear it. 0 Transmit pending or in progress 1 Transmit complete
4 ETC	Early transmit complete. Indicates the SIM transmitter has finished sending the current byte and the transmit FIFO is empty. This bit differs from TC in that it is set before the guard time of the last byte has elapsed. ETC generates an interrupt if SIM_IMR[ETCIM] is cleared. Write one to ETC to clear it. 0 Transmit pending or in progress 1 Transmit complete
3 TFE	Transmit FIFO empty. Indicates the transmit FIFO is empty. This bit is set when the last byte in the transmit FIFO has been transferred to the SIM transmit shift register. TFE generates an interrupt if SIM_IMR[TFEIM] is cleared. Write one to TFE to clear it. 0 Transmit FIFO is not empty 1 Transmit FIFO is empty
2–1	Reserved, must be cleared
0 XTE	Transmit NACK threshold error. Indicates the transmit NACK threshold has been reached. When XTE is set, no further transmissions are done until XTE is cleared. Any data transmissions pending in the transmit FIFO are aborted, and TC, ETC, and TFE are set. XTE generates an interrupt if SIM_IMR[XTM] is cleared. Write one to XTE to clear it. 0 Transmit NACK threshold has not been reached 1 Transmit NACK threshold reached; transmitter frozen

26.4.11 SIM Receive Status Register (SIM_RSR)

0xFC0A_C02C (SIM_RSR) 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	BGT	BWT	RTE	CWT	CRC OK	LRC OK	RDRF	RFD	0	0	0	OEF
W					w1c	w1c	w1c	w1c								w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 26-12. SIM Receive Status Register (SIM_RSR)

Table 26-13. SIM_RSR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11 BGT	Block guard time error flag. Indicates the block guard time was too small. The threshold is set by the block guard time register (SIM_BGT). 0 Block guard time was sufficient 1 Block guard time was too small
10 BWT	Block wait time error flag. Indicates the block wait time has been exceeded. The threshold is set by the block wait time registers (SIM_BWT, SIM_BWTH). 0 Block wait time not exceeded 1 Block wait time was exceeded
9 RTE	Receive NACK threshold error flag. Indicates if the number of consecutive NACK's generated by the SIM module in response to receive parity errors for the byte being received, equals the value programmed in SIM_RTHR[RTH]. This bit never sets unless SIM_CR[ANACK] is set. SIM_CR*n*[SAPD] must be set to enable the threshold error to trigger the auto power down sequence. Write one to RTE to clear it. Clearing this bit resets the internal counter for consecutive NACK's being transmitted for a given byte. 0 Number of NACKs generated by the receiver is less than the value programmed in SIM_RTHR[RTH] 1 Number of NACKs generated by the receiver is equal to the value programmed in SIM_RTHR[RTH]
8 CWT	Character wait time counter flag. Indicates when the time between received characters is equal to or greater than the value programmed in the SIM_CWTR register. 0 No CWT violation has occurred 1 Time between two consecutive characters exceeded the value in SIM_CWTR
7 CRCOK	Cyclic redundancy check okay flag. Indicates when the calculated 16-bit CRC value matches the expected value for the current input data stream. The value is calculated across all received characters from the point SIM_CR[CRCEN] is set. The current CRC residual is reset by three mechanisms: • Clear SIM_CR[CRCEN] • Set SIM_EN[TXEN] • Automatically by hardware when ETC flag is set at the end of a transmission. 0 Current CRC value does not match remainder 1 Current calculated CRC value matches the expected result
6 LRCOK	Linear redundancy check okay flag. Indicates when the calculated 8-bit LRC value is zero value for the current input data stream. The value is calculated across all received characters from the point SIM_CR[LRCEN] is set. The current LRC residual is reset by three mechanisms: • Clear SIM_CR[LRCEN] • Set SIM_EN[TXEN • Automatically by hardware when ETC flag is set at the end of a transmission. 0 Current LRC value does not match remainder 1 Current calculated LRC value matches the expected result of zero
5 RDRF	Receive data register full. Indicates the SIM receive FIFO has reached the threshold level set by SIM_RTHR[RDT]. RDRF is set any time the number of unread bytes in the receive FIFO is equal to or greater than the value set by RDT. The flag is cleared by reading enough bytes out of the receive FIFO to bring the number of bytes left in the FIFO below RDT level. Another way to clear the flag is to set RDT higher than the number of unread bytes currently in the FIFO. RDRF generates an interrupt if SIM_CR[RIM] is cleared. 0 Number of unread bytes in receive buffer is less than value set by RDT 1 Number of unread bytes in receive buffer is greater than or equal to the value set by RDT

Table 26-13. SIM_RSR Field Descriptions (continued)

Field	Description
4 RFD	Receive FIFO has unread data. Indicates there is at least one unread byte in the receive data FIFO. This bit is only cleared by reading all bytes out of the receive FIFO. RFD cannot be used to generate an interrupt. Normally, the SIM triggers the interrupt with RDRF and software uses RFD to read all of the bytes out of the receive FIFO. 0 There are no unread bytes in the receive FIFO 1 There is at least one unread byte in the receive FIFO
3–1	Reserved, must be cleared.
0 OEF	Overrun error flag. Indicates the SIM was unable to store received data due to already having 285 unread bytes in the FIFO. It does not necessarily indicate that data has been lost. If SIM_CR[ONACK] is set, a NACK pulse is generated on bytes that would otherwise cause a loss of data due to a full FIFO. These bytes should be retransmitted by the SIM card which implies that no data has actually been lost. In this case, the OEF flag is just an indicator that this situation has occurred which may be helpful in system debug. If ONACK is not set, a set OEF flag indicates a loss of data since all bytes received with the OEF flag set will indeed be lost (including the byte that caused the bit to set). The OEF flag generates an interrupt if SIM_IMR[OIM] is cleared. The OEF flag is a write-one-to-clear bit. 0 No overrun error has occurred 1 A byte was received when the received FIFO was already full

26.4.12 SIM Interrupt Mask Register (SIM_IMR)

Address: 0xFC0A_C030 (INT_MASK) 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	BGTM	BWTM	RTM	CWTM	GPCNTM	TDTFM	TFOM	XTM	TFEIM	ETCIM	OIM	TCIM	RIM
W																
Reset	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 26-13. SIM Interrupt Mask Register (SIM_IMR)

Table 26-14. SIM_IMR Field Descriptions

Field	Description
31–13	Reserved, must be cleared.
12 BGTM	Block guard time interrupt mask. Enables the interrupt mask for SIM_RSR[BGT]. 0 Enabled 1 Masked

Table 26-14. SIM_IMR Field Descriptions (continued)

Field	Description
11 BWTM	Block wait time interrupt mask. Enables the interrupt mask for SIM_RSR[BWT]. 0 Enabled 1 Masked
10 RTM	Receive NACK threshold interrupt mask. Enables the interrupt mask for SIM_RSR[RTE]. 0 Enabled 1 Masked
9 CWTM	Character wait time interrupt mask. Enables the interrupt mask for SIM_RSR[CWT]. 0 Enabled 1 Masked
8 GPCNTM	General purpose counter interrupt mask. Enables the interrupt mask for SIM_TSR[GPCNT]. 0 Enabled 1 Masked
7 TDTFM	Transmit data threshold interrupt mask. Enables the interrupt mask for SIM_TSR[TDTF]. 0 Enabled 1 Masked
6 TFOM	Transmit FIFO overfill error interrupt mask. Enables the interrupt mask for SIM_TSR[TFO]. 0 Enabled 1 Masked
5 XTM	Transmit threshold interrupt mask. Enables the interrupt mask for SIM_TSR[XTE]. 0 Enabled 1 Masked
4 TFEIM	Transmit FIFO empty interrupt mask. Enables the interrupt mask for SIM_TSR[TFE]. 0 Enabled 1 Masked
3 ETCIM	Early transmit complete interrupt mask. Enables the interrupt mask for SIM_TSR[ETC]. 0 Enabled 1 Masked
2 OIM	Overrun interrupt mask. Enables the interrupt mask for SIM_RSR[OEF]. 0 Enabled 1 Masked
1 TCIM	Transmit complete interrupt mask. Enables the interrupt mask for SIM_TSR[TC]. 0 Enabled 1 Masked
0 RIM	Receive Interrupt Mask. Enables the interrupt mask for SIM_RSR[RDRF]. 0 Enabled 1 Masked

26.4.13 SIM Data Format Register (SIM_FORMAT)

Address: 0xFC0A_C040 (SIM_FORMAT) 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	IC
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-14. SIM Data Format Register (SIM_FORMAT)

Table 26-15. SIM_FORMAT Field Descriptions

Field	Description
31–1	Reserved, must be cleared
0 IC	Inverse convention. Configures the SIM to use inverse or direct convention for its data format. This bit can be controlled by software, but it is normally set by hardware as a result of the interpretation of the initial character when in ICM mode. 0 Direction convention transfers enabled (default). 1 Inverse convention transfers enabled.

26.4.14 SIM Transmit Threshold Register (SIM_TTHR)

Address: 0xFC0A_C044 (SIM_TTHR) 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	XTH				TDT			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-15. SIM Transmit Threshold Register (SIM_TTHR)

Table 26-16. SIM_TTHR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.

Table 26-16. SIM_TTHR Field Descriptions (continued)

Field	Description
7–4 XTH	Transmit NACK threshold. When this threshold is reached for the current byte being transmitted, SIM_TSR[XTE] is set. This causes the remaining transmissions queued in the transmit FIFO to be aborted and no more transmissions occur until software clears XTE. To trigger XTE, a given byte being transmitted must reach the XTH threshold itself. Transmit NACKs accumulated on one byte are not carried over to the next. 0x0 XTE is never set; retransmission after NACK reception is disabled 0x1 XTE is set after 1 NACK is received; 0 retransmissions occur 0x2 XTE is set after 2 NACKs are received; at most 1 retransmission occurs 0x3 XTE is set after 3 NACKs are received; at most 2 retransmissions occur ... 0xF XTE is set after 15 NACKs are received; at most 14 retransmissions occur
3–0 TDT	Transmit data threshold. When the number of bytes in the transmit FIFO is less than or equal to TDT, SIM_TSR[TDTF] is set.

26.4.15 SIM Transmit Guard Control Register (SIM_TGCR)

Address: 0xFC0A_C048 (SIM_TGCR) 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	RCVR 11	GETU							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-16. SIM Transmit Guard Control Register (SIM_TGCR)

Table 26-17. SIM_TGCR Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8 RCVR11	Receiver 11 ETUs. Configure the SIM receiver for 11 ETU operation (1 stop bit). This bit is provided for the T=1 protocol. 0 Receiver configured for 12 ETU operation 1 Receiver configured for 11 ETU operation
7–0 GETU	Transmit guard ETUs. Controls the number of additional elementary time units (ETUs) inserted between bytes transmitted by the SIM. An ETU is equivalent to one bit time at the given baud rate (for example, the length of a start bit). The guard time has no effect on the SIM receiver. 0x00 No additional ETUs inserted ... 0xFE 254 additional ETUs inserted 0xFF One ETU removed (Stop bits reduced from two to one)

26.4.16 SIM Open Drain Configuration Control Register (SIM_ODCR)

Address: 0xFC0A_C04C (SIM_ODCR) 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	OD P1	OD P0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-17. SIM Open Drain Configuration Control Register (SIM_ODCR)

Table 26-18. SIM_ODCR Field Descriptions

Field	Description
31–2	Reserved
1 ODP1	Open drain control for port 1. Controls whether the transmit pin on port 1 is open-drain. If SIM_SETUP[AMODE] is set, this bit has no effect. 0 XMT pin on port 1 is push-pull 1 XMT pin on port 1 is open-drain
0 ODP0	Open drain control for port 0. Controls whether the transmit pin on port 0 is open-drain. 0 XMT pin on port 0 is push-pull 1 XMT pin on port 0 is open-drain

26.4.17 SIM Reset Control Register (SIM_RCR)

Address: 0xFC0A_C050 (SIM_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	0	0	0	0	DBUG	STOP	DOZE	KILL CLOCK	0	FLUSH XMT	FLUSH RCV
W														SOFT RST		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-18. SIM Reset Control Register (SIM_RCR)

Table 26-19. SIM_RCR Field Descriptions

Field	Description
31–7	Reserved, must be cleared.
6 DBUG	Debug. Configures the SIM when a debug event occurs. When set, a debug event causes the receive FIFO read pointer to freeze. 0 Debug event has no affect on SIM module 1 Debug event prohibits read pointer changes for receive FIFO
5 STOP	Stop. Configures the SIM when a processor stop instruction is executed. This bit provides support for SIM cards that do not allow the SIM card clock to be stopped while power is applied. 0 Stop instruction shuts down all SIM clocks 1 Stop instruction shuts down all clocks except for the BAUD_CLK (clock provided to SIM Card)
4 DOZE	Doze. Configures the SIM module when a processor DOZE instruction is executed.0DOZE instruction has no effect on SIM module 1 DOZE instruction will cause SIM module to gate SIM clocks when the transmit FIFO is empty
3 KILL_CLOCK	Kill SIM Clock. Disables the SIM clock input to the SIM module. This bit gates all SIM clocks including the SIM card clock regardless of the state of the STOP bit. 0 SIM input clock enabled 1 SIM input clock disabled
2 SOFT_RST	Software reset. Resets the entire SIM module. This acts the same as a hardware reset for the SIM module. This bit is self-clearing. **Note:** Software should allow a minimum of 4 reference clock cycles (CKIH) before attempting to access the SIM module after a software reset. 0 SIM normal operation 1 SIM held in reset
1 FLUSH_XMT	Flush transmitter. Resets the SIM transmitter. The receive portion of the SIM module is not affected. The software must clear this bit before the SIM transmitter can operate. 0 SIM transmitter normal operation 1 SIM transmitter held in reset
0 FLUSH_RCV	Flush Receiver. Resets the SIM receiver. The transmit portion of the SIM module is not affected. The software must clear this bit before the SIM receiver can operate. 0 SIM receiver normal operation 1 SIM receiver held in reset

26.4.18 SIM Character Wait Time Register (SIM_CWTR)

Address: 0xFC0A_C054 (SIM_CWTR) 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	CWT															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 26-19. SIM Character Wait Time Register (SIM_CWTR)

Table 26-20. SIM_CWTR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 CWT	Character wait time. Specifies the number of ETU times allowed between characters. Default is 0xFFFF

26.4.19 SIM General Purpose Counter Register (SIM_GPCNT)

Address: 0xFC0A_C058 (SIM_GPCNT) 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	GPCNT															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 26-20. SIM General Purpose Counter Register (SIM_GPCNT)

Table 26-21. SIM_GPCNT Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 GPCNT	General purpose counter. The value written to this field is compared to the general purpose counter in the SIM module. Once the general purpose counter reaches this value, SIM_TSR[GPCNT] is set. This counter is intended to be used for any events that must be monitored for duration based on the card clock, receiver sample rate, or ETU rate (transmit clock). Example: ATR arrival time and ATR duration. Default is 0xFFFF

26.4.20 SIM Divisor Register (SIM_DIV)

Address: 0xFC0A_C05C (SIM_DIV) 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	DIVISOR							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 26-21. SIM Divisor Register (SIM_DIV)

Table 26-22. SIM_DIV Field Descriptions

Field	Description
31–8	Reserved
7–0 DIVISOR	Divisor register. The value written to this register generates the SIM receive clock. SIM_CR[BAUD_SEL] must be set to 111 to control the divisor value using the DIVISOR register. Default is 0xFF

26.4.21 SIM Block Wait Time Low Register (SIM_BWTL)

Address: 0xFC0A_C060 (SIM_BWTL) 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	BWT															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 26-22. SIM Block Wait Time Low Register (SIM_BWTL)

Table 26-23. SIM_BWTL Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BWT	Lower block wait time. This value is the lower 16 bits of the block wait time. The time from the start bit of the last byte sent from the SIM to the start bit of the first byte sent from the smart card must be less than the 32-bit value formed SIM_BWTH and SIM_BWTL. If not, SIM_RSR[BWT] is set. Default is 0xFFFF

26.4.22 SIM Block Guard Time Register (SIM_BGT)

Address: 0xFC0A_C064 (SIM_BGT) 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	BGT															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-23. SIM Block Guard Time Register (SIM_BGT)

Table 26-24. SIM_BGT Field Descriptions

Field	Description
31–16	Reserved, must be cleared
15–0 BGT	Block guard time. The time from the start bit of the last byte sent from the SIM to the start bit of the first byte sent from the smart card must be greater than this value. If not, SIM_RSR[BGT] is set. Default is 0x0000

26.4.23 SIM Block Wait Time High Register (SIM_BWTH)

Address: 0xFC0A_C068 (SIM_BWTH) 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	BWT															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 26-24. SIM Block Wait Time High Register (SIM_BWTH)

Table 26-25. SIM_BWTH Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BWT	Upper block wait time. This value is the upper 16 bits of the block wait time. The time from the start bit of the last byte sent from the SIM to the start bit of the first byte sent from the smart card must be less than the 32-bit value formed SIM_BWTH and SIM_BWTL. If not, SIM_RSR[BWT] is set. Default is 0xFFFF

26.4.24 SIM Transmit FIFO Status Register (SIM_TFSR)

Address: 0xFC0A_C06C (SIM_TFSR) 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	CNT				WPTR				RPTR			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-25. SIM Transmit FIFO Status Register (SIM_TFSR)

Table 26-26. SIM_TFSR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11–8 CNT	Indicates the number of bytes in the transmit FIFO. 0000 FIFO is empty or full 0001 One byte in the FIFO ... 1111 15 bytes in the FIFO
7–4 WPTR	Indicates the transmit FIFO write pointer.
3–0 RPTR	Indicates the transmit FIFO read pointer.

26.4.25 SIM Receive FIFO Counter Register (SIM_RFCR)

Address: 0xFC0A_C070 (SIM_RFCR) 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	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	0	0	0	0	0	0

Figure 26-26. SIM Receive FIFO Counter Register (SIM_RFCR)

Table 26-27. SIM_RFCR Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8–0 CNT	Indicates the number of bytes that can be written into the receive FIFO. A value of zero indicate the receive FIFO is empty or full.

26.4.26 SIM Receive FIFO Write Pointer Register (SIM_RFWP)

Address: 0xFC0A_C074 (SIM_RFWP) 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	WPTR								
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-27. SIM Receive FIFO Write Pointer Register (SIM_RFWP)

Table 26-28. SIM_RFWP Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8–0 WPTR	Indicates the receive FIFO write pointer.

26.4.27 SIM Receive FIFO Read Pointer Register (SIM_RFRP)

Address: 0xFC0A_C078 (SIM_RFRP) 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	RPTR								
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-28. SIM Receive FIFO Read Pointer Register (SIM_RFRP)

Table 26-29. SIM_RFRP Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8–0 RPTR	Indicates the receiver FIFO read pointer.

26.5 Functional Description

To best describe the organization of the SIM module from a user's point of view, it is instructive to view the SIM at a number of different levels of hierarchy. The SIM module is essentially a standard UART with some special provisions made for SIM card communication. The SIM consists of seven main parts:

- Clock generator
- Transmitter
- Receiver
- Port controller
- General purpose counter
- LRC blocks
- CRC blocks

26.5.1 SIM Clock Generator

The clock generator is responsible for generating the baud rate clock (BAUD_CLK), and clocks to the transmitter, receiver, and port controller sections of the SIM module.

26.5.1.1 Baud Clock Generation

The baud rate clock generation performed by the clock generator results in different frequencies. The default frequency is a divide by two of the peripheral clock. The baud rate can be programmed to be another divisor using the SIM_PRE register as shown in Section 26.4.7, "SIM Clock Prescaler Register (SIM_PRE)". The baud clock is generated in two forms in the design. The BAUD_CLK that is used by the SIM cards (SCLK0, SCLK1) must be 45–55% duty cycle at the divide values. This is necessary to meet the requirements of the ISO 7816 specification. The BAUD_CLK used internal to the SIM module is a gated version of the peripheral clock.

26.5.1.2 Transmitter Clock Generation

The transmitter clock (XMT_CLK) is generated by the clock generator based on the values passed to it for the baud rate select (baud_sel[2:0]) and sample12. The transmit clock is gated by the transmit enable (TXEN) register bit. When the transmitter is enabled, the clock generator counts the appropriate number of receive clock (rcv_clk) positive edges to determine when to toggle the transmitter clock output. The transmitter clock is always based upon the receive clock.

26.5.1.3 Receiver Clock Generation

The receiver clock (RCV_CLK) is generated by the clock generator based on the value passed to it for the baud rate select (SIM_CR[BAUD_SEL]). The receiver clock is gated by the receiver enable (SIM_EN[RXEN]) register bit. When the receiver is enabled, the clock generator counts the appropriate number of BAUD_CLK positive edges to determine when to toggle the receiver clock output. The number of BAUD_CLK positive edges is set by SIM_CR[BAUD_SEL] and is programmable to these divisors: 31 (slowest because used with the 372/1 Fi/Di rate), 32, 16, 8, 4, 2, 1, and SIM_DIV[DIVISOR].

26.5.1.4 Port Control Clock Generation

The port controller clocks are provided by the clock generator for the SIM card ports. These clocks are equivalent in frequency to the BAUD_CLK and are gated by the SIM clock enable (SIM_CR*n*[SCEN]) signals. The level at which the card clocks (SIM_CLK*n*) are stopped when disabled is determined by the SIM clock select polarity (SIM_CR*n*[SCSP]) inputs to the clock generator. Synchronizers are implemented to ensure glitch free operation of the card clocks when enabling, disabling, or changing the clock stopped polarity.

26.5.1.5 Low Power Mode Clock Control

The clock generator block is responsible for gating the clocks to the SIM module appropriately whenever a low power mode instruction is decoded. There aretwo available low-power states of the processor: stop and doze. The response to doze mode is controlled by SIM_RCR[DOZE]. Likewise, the response to stop mode is controlled by SIM_RCR[STOP]. See Section 26.4.17, “SIM Reset Control Register (SIM_RCR)”.

26.5.2 SIM Transmitter

The transmitter block comprises the following sections of logic: transmit state machine, transmit shift register, transmit FIFO, guard time generator, transmit NACK control, and transmit data convention.

26.5.2.1 Transmit State Machine

The Transmit state machine is the heart of the transmitter block. The state machine is responsible for sequencing through a transmit operation while reacting to inputs from the receiver, the transmit FIFO, and the guard time circuit. See Figure 26-29 for flow diagram of the transmit state machine.

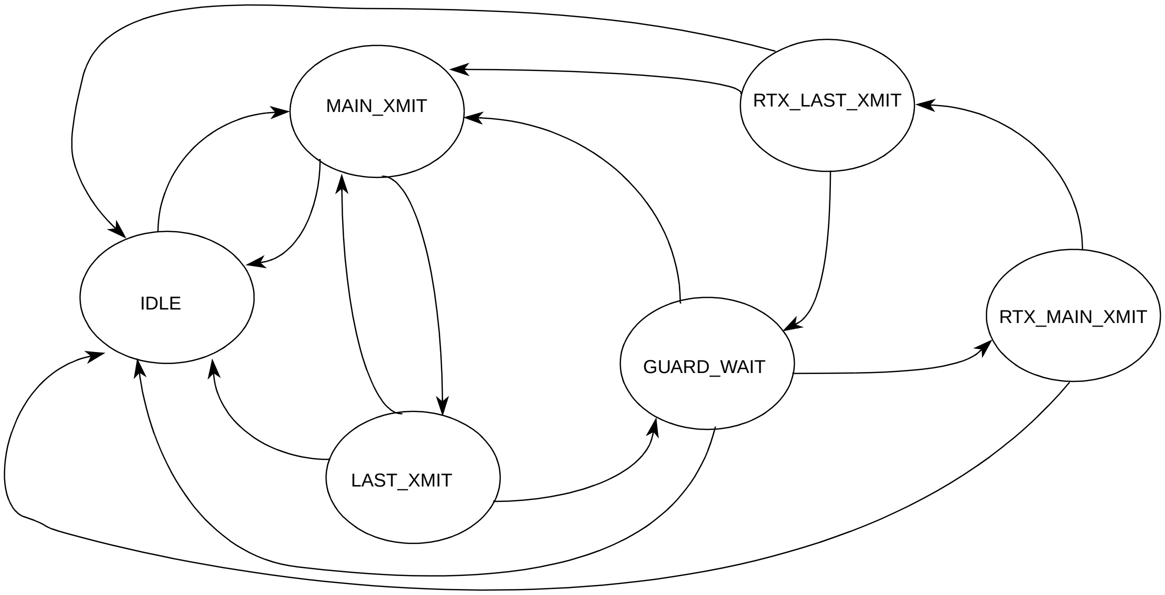


Figure 26-29. Transmit State Machine

The functions performed by each state are:

- IDLE
  - This is the initial state. The state machine waits here until SIM_EN[TXEN] is set and a write to the transmit FIFO has occurred. The data pointed to by the transmit read pointer is loaded into the shift register, and the state machine transitions to the MAIN_XMIT state. Any time SIM_EN[TXEN] is cleared, the state machine returns to this state.
- MAIN_XMIT
  - The transmitter is operating normally in this state. The data in the shift register is shifting once every transmit clock cycle. When the second to last bit of the current transmission is about to be sent, the state machine transitions to the LAST_XMIT state.
- LAST_XMIT
  - This state transmits the last bit of the current transmission and determines the next operation. One of the following will occur.
    - If SIM_TGCR[GETU] is non-zero, jump to the GUARD_WAIT state.
    - If a transmit NACK error occurred, with a zero in SIM_TGCR[GETU], jump to MAIN_XMIT state to retransmit the current byte.
    - If no transmit NACK, and SIM_TGCR[GETU] is zero, load the shift register, jump to MAIN_XMIT to transmit the next byte
    - If no transmit NACK, SIM_TGCR[GETU] is zero, and the FIFO is empty, jump to IDLE state; set the transmit complete (TC) flag.
- GUARD_WAIT
  - The state machine remains in this state until the guard time counter has expired.
    - If a transmit NACK error occurred on last transmission, jump to RTX_MAIN_XMIT and re-transmit

    - If no transmit NACK and the FIFO is not empty, load the shift register, jump to MAIN_XMIT to transmit the next byte.
    - If no transmit NACK and the FIFO is empty, return to the IDLE state.
    - If transmit NACK threshold is detected, stop transmitter, set XTE flag, and jump to the IDLE state.
- RTX_MAIN_XMIT
  - The transmitter is operating normally in this state. The data in the shift register is shifting once every transmit clock cycle. When the second to last bit of the current transmission is about to be sent, the state machine transitions to the RTX_LAST_XMIT state. This state is identical to the MAIN_XMIT state except that it retransmits the previously NACKed byte.
- RTX_LAST_XMIT
  - This state transmits the last bit of the current re-transmission and determines the next operation. One of the following will occur.
    - If SIM_TGCR[GETU] is non-zero, jump to the GUARD_WAIT state.
    - If a transmit NACK error occurred, with a zero in SIM_TGCR[GETU], jump to GUARD_WAIT state to check transmit NACK threshold.
    - If no transmit NACK, SIM_TGCR[GETU] is zero, and the FIFO is not empty, load the shift register, jump to MAIN_XMIT to transmit the next byte
    - If no transmit NACK, SIM_TGCR[GETU] is zero, and the FIFO is empty, jump to IDLE state; set the transmit complete (TC) flag.

26.5.2.2 Transmit Shift Register

The transmit shift register is 11 bits wide and controlled by the transmit state machine described previously. The shift register shifts out data at the transmit clock frequency (xmt_clk).

26.5.2.3 Transmit FIFO

The transmit FIFO is implemented inside the transmitter block. The FIFO depth is 16 bytes. The FIFO block is shared by both SIM module ports. The transmit FIFO cannot be accessed by the alternate SIM module through the alternate port. Each write to the transmit FIFO increases the transmit FIFO write pointer. Each time the transmit shift register is loaded from the transmit FIFO, the transmit FIFO read pointer is incremented. When the read and write pointers are equal, the transmit FIFO empty flag (TFE) is set. Software has no visibility of the transmit FIFO pointers, but a transmit FIFO threshold value can be set to alert the software when the number of bytes in the FIFO has reached a specified level. A read of the transmit FIFO register (SIM_TBUF) returns the last byte written to the FIFO. Writes to the transmit FIFO can occur at any time.

The transmit FIFO can be flushed by setting SIM_RCR[FLUSHTX]. A transmit NACK threshold error (XTE) halts the transmitter and flushes the transmit FIFO. The flush operation resets the transmit read and write pointers to the same values. Everything in the transmitter block is reset by the transmit flush operation. This does not include the control registers associated with the transmitter. The transmit data threshold flag (TDTF) is not cleared by the flush operation.

26.5.2.4 Transmit Guard Time Generator

The guard time generator is simply a counter that is clocked by the transmit clock in order to delay the beginning of the next transmission and the setting of the transmit complete interrupt flag (TC) by a programmable amount of transmit bit widths (ETUs). The duration of the count is controlled by SIM_TGCR[GETU]. See Figure 26-30 for depiction of the three transmit operations to show the effect of the guard time generator logic

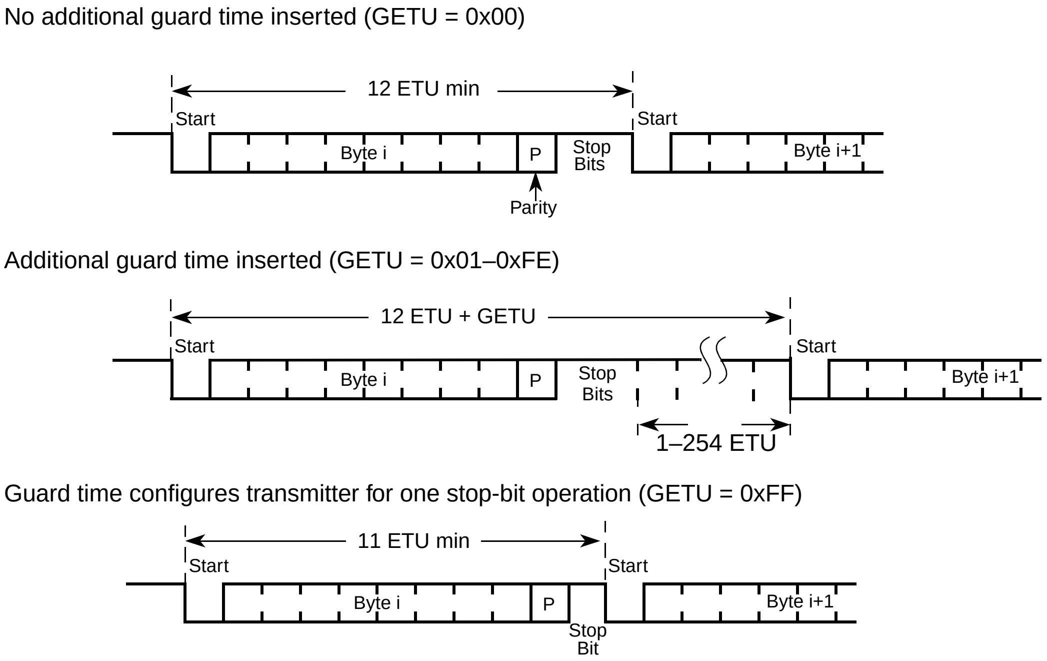

No additional guard time inserted (GETU = 0x00)
12 ETU min
Start
Byte i
P
Stop
Bits
Start
Byte i+1
Parity
Additional guard time inserted (GETU = 0x01–0xFE)
12 ETU + GETU
Start
Byte i
P
Stop
Bits
Start
Byte i+1
1–254 ETU
Guard time configures transmitter for one stop-bit operation (GETU = 0xFF)
11 ETU min
Start
Byte i
P
Stop
Bit
Start
Byte i+1

Figure 26-30. Transmit Guard Time

26.5.2.5 Transmit NACK Generator

The transmit NACK generator is responsible for driving the transmitter output low during the stop bit time to signify an error was detected in the received data from the SIM card. This logic responds to a NACK request generated by the receiver block. Figure 26-31 shows a typical SIM transaction with the NACK pulse inserted.

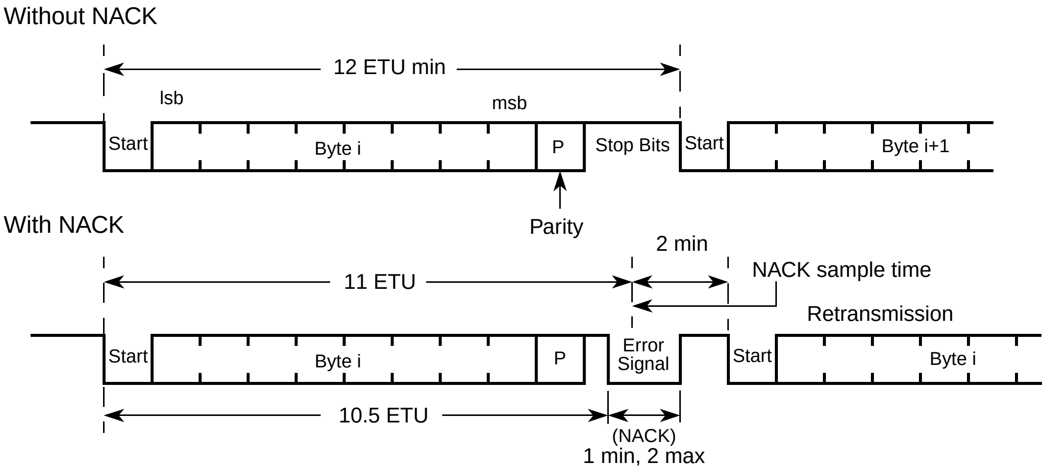
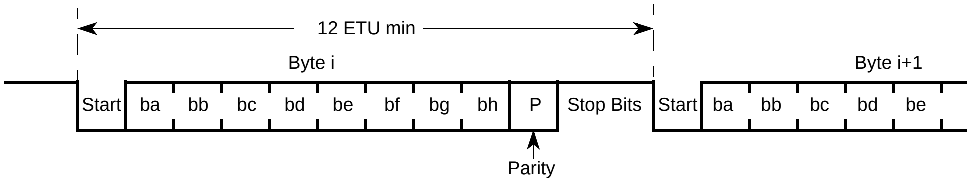


Figure 26-31. Transmit NACK Generator

The transmit NACK generator is also responsible for keeping track of the number of NACKs received during a transmit operation. The SIM receive state machine detects NACKs generated by the SIM card, and reports them to the transmit NACK logic. When the number of detected NACKs equals the programmed threshold (SIM_TTHR[XTH]), an interrupt is generated, the transmit FIFO is flushed, and the transmitter is disabled.

26.5.2.6 Transmit Data Convention Logic

The transmit data convention logic supports the two different data conventions available in SIM cards. See Figure 26-32 for illustration of SIM data conventions.



Parity bit: If configured for even parity, total number of logic ones in the 9 bits (8 bits data, 1 parity bit) is even
if configured for odd parity, total number of logic ones in the 9 bits (8 bits data, 1 parity bit) is odd
When configured for inverse convention, the parity bit is inverted by the SIM before being transmitted.

Direct convention: ba is lsb of the data byte sent. bh is msb.
Neither the data bits nor parity bit is logically inverted.

Inverse convention: ba is msb of the data byte sent. bh is lsb.
Both the data bits and parity bit are logically inverted by hardware.

Figure 26-32. SIM Data Conventions

The direct data convention is the default. If SIM_FORMAT[IC] is set, the transmit data convention logic converts the output of the transmit FIFO to the inverse convention before sending it to the transmit shift register.

26.5.3 SIM Receiver

The receiver block is comprised of the receive state machine and the receive FIFO.

26.5.3.1 Receive State Machine

The receive state machine samples the receive data pin and captures the bit value into the receive shift register. Additionally, the receive state machine detects the start bit, parity errors, framing errors, and the initial character when operating in initial character mode.

Once enabled by SIM_EN[RXEN] , the receive state machine sequences through the states as shown in Figure 26-33.

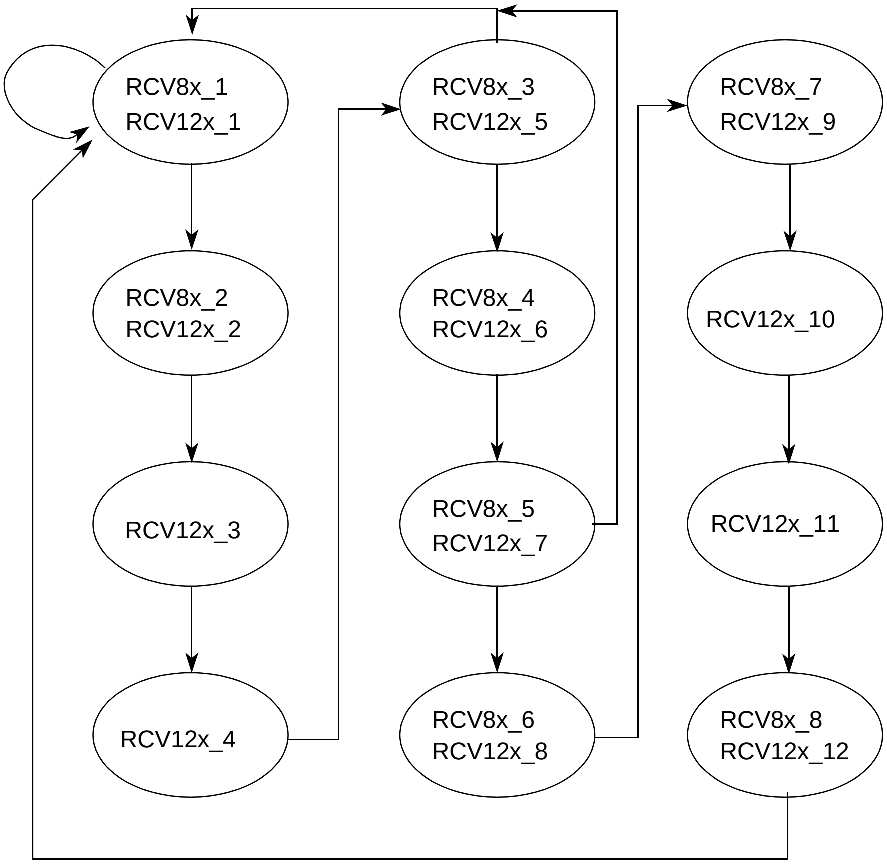


Figure 26-33. Receive State Machine

The states identified with an 8x are used when operating in a 8x oversampling mode. The states identified with a 12x are used when operating in a 12x oversampling mode (SIM_CR[SAMPLE12] is set). The number following the oversampling mode identifier represents the state number in the current mode. There are 12 states in 12x mode, and eight states in 8x mode. Some states simply implement a one RCV_CLK delay. States that perform additional functions are:

- RCV8x_1, RCV12x_1
  - This is the initial state of the receive state machine. If the first bit has not been received, the state machine remains in this state until a valid start bit is detected. For every subsequent bit, this state is simply a one RCV_CLK cycle delay.
- RCV8x_2, RCV12x_2
  - This state captures the first sample of the current receive data input.

- RCV12x_3
  - This state captures the second sample of the current receive data input.
- RCV12x_4
  - This state captures the third sample of the current receive data input.
- RCV8x_3, RCV12x_5
  - This state checks if the SIM is receiving a correct start Bit. If not, return to state 1.
- RCV8x_4, RCV12x_6
  - If in the 11th bit, check for parity or initial character errors and send NACK if needed.
- RCV8x_5, RCV12x_7
  - Store current bit value in the shift register. If this is the first bit and the value is not zero, restart the state machine.
- RCV8x_6, RCV12x_8
  - If the current bit is the last bit of the transfer, set flag to transfer shift register to receive buffer.
- RCV8x_7, RCV12x_9
  - Clear flag for transferring shift register to receive buffer.
- RCV12x_10
  - If the current bit is the last bit of the transfer (first stop bit), this state samples the NACK window.
- RCV12x_11
  - If the current bit is the last bit of the transfer (first stop bit), this state samples the NACK window.
- RCV8x_8, RCV12x_12
  - This state represents the end of the current receive input bit. Several operations occur during this state, including:
    - Increment bit counter
    - Perform a majority vote on the NACK samples and notify the transmitter if a NACK pulse was detected.

26.5.3.2 Data Sampling/Voting

The receive state machine runs at the receive clock rate (RCV_CLK). This clock oversamples the received data at either an 8x or 12x sample rate. For each input bit, the receive state machine captures three samples. A majority voting algorithm is then applied to determine the value of the bit received. The value common to two or more samples is determined to be the bit value in the receive shift register.

26.5.3.3 Start Bit Detection

The SIM receive input is defined as high when not active. The data transmission is defined to begin with a low pulse for a bit duration. This is called the start bit. The receive state machine is responsible for detecting and validating the start bit. The receive state machine samples the start bit three times using a majority voting scheme to determine if the start bit is valid. This will effectively filter out any low receive inputs shorter than one RCV_CLK period. See Figure 26-34 for illustration of a typical SIM data transaction with the start bit identified.

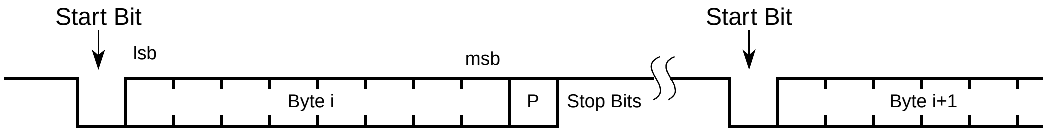


Figure 26-34. Start Bit Diagram

26.5.3.4 Parity Error Detection

The receive state machine is responsible for detecting parity errors in the received data. Data is always transmitted with even parity, except when in inverse convention mode. In this mode, all data and parity bits are complemented making the data appear as odd parity. The parity bit is defined as the tenth bit of the received data. The parity of the second through tenth received bits is calculated by the receiver parity logic. This logic determines if the parity of the nine received bits is correct. See Figure 26-35 for illustration of a typical SIM data transaction with the parity bit identified.

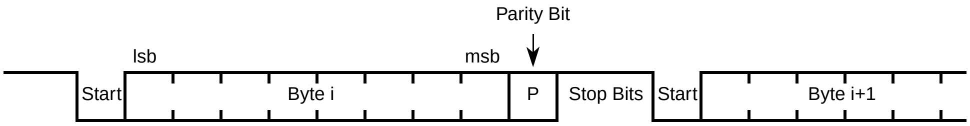


Figure 26-35. Parity Bit Diagram

When a parity error is detected on a given byte, SIM_RBUF*n*[PE] is set for that byte. A parity error cannot generate an interrupt. However, it can signal the SIM transmitter to send a NACK pulse to the SIM card asking for a retransmission of the corrupted data. NACK generation upon a parity error is enabled by setting SIM_CR[ANACK].

26.5.3.5 Framing Error Detection

The receive state machine is responsible for detecting framing errors in the received data. A SIM data transaction is defined as 11 or 12 bits long consisting of the start bit, eight data bits, one parity bit, and one or two stop bits. A framing error occurs when the stop bit is not detected during the 11th bit time of a data transaction. The stop bit is generally defined as two bit times (ETUs) of a high pulse following the parity bit. When SIM_TGCR[GETU] is 0xFF, the stop bit is defined as one bit time. A framing error occurs only when the parity bit of the current byte is low, and the stop bit arrives late. See Figure 26-36 for illustration of a typical SIM data transaction with the stop bits identified. Also shown is a SIM data transaction with a late arriving stop bit indicating a framing error.

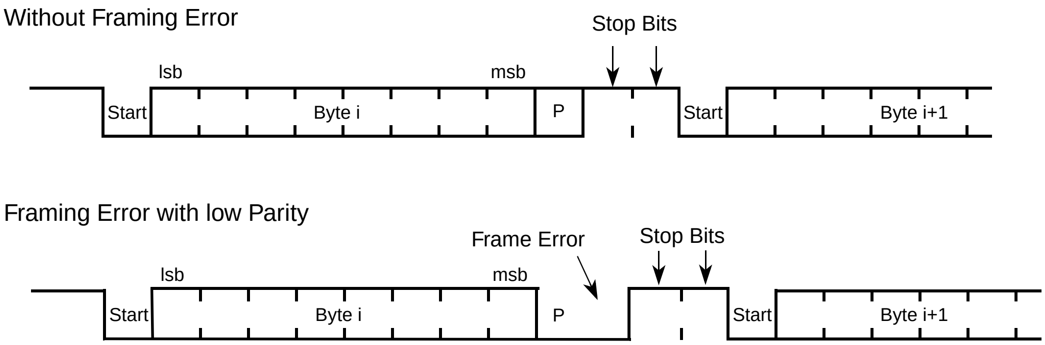


Figure 26-36. Framing Error Diagram

When a framing error is detected on a given byte, SIM_RBUF*n*[FE] is set for that byte. A framing error cannot generate an interrupt, nor can it create a NACK pulse to the SIM card asking for a retransmission of the corrupted data.

26.5.3.6 NACK Detection

The existence of the NACK pulse is sampled by the receive state machine at 11 elementary time units (ETUs) after the falling edge of the start bit. An ETU is equivalent in time to one transmit clock period. When the receiver detects a NACK, it signals the transmitter that an error occurred. The transmitter will not initiate retransmission for at least another two ETU times as required by the ISO 7816 specification.

26.5.3.7 Initial Character Detection

The SIM receive state machine supports the detection of special characters that allow it to determine what data format is being used by the connected SIM card. When placed in initial character mode, the SIM expects to receive one of two potential characters that it will use to set the data format control bit (SIM_FORMAT[IC]).

The two possible data formats are inverse convention and direct convention. Figure 26-37 and Figure 26-38 illustrate the differences between the two formats. Essentially, inverse convention flips the order of the data and the data and parity bits are logically inverted. When receiving inverse convention data, the transformation of the data back to direct convention format is done in hardware, including the inversion of the data and parity bits.

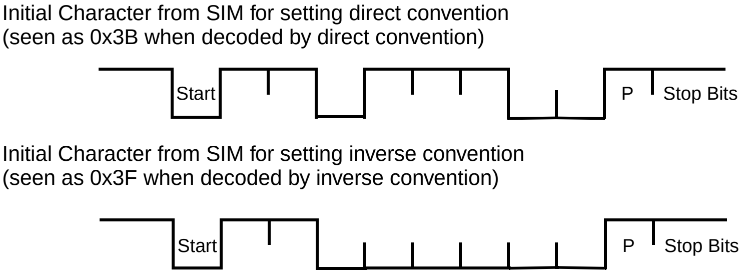
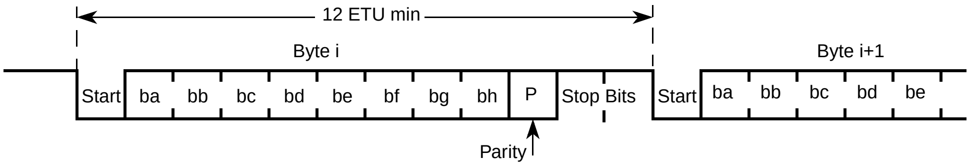

Initial Character from SIM for setting direct convention
(seen as 0x3B when decoded by direct convention)
Start
P
Stop Bits
Initial Character from SIM for setting inverse convention
(seen as 0x3F when decoded by inverse convention)
Start
P
Stop Bits

Figure 26-37. Valid Initial Characters



Parity bit: If configured for even parity, total number of logic ones in the 9 bits (8 bits data, 1 parity bit) will be even.
If configured for odd parity, total number of logic ones in the 9 bits (8 bits data, 1 parity bit) will be odd.
When configured for inverse convention, the parity bit is inverted by SIM card before being transmitted.

Direct convention: ba is lsb of data byte to be sent. bh is msb.
Neither the data bits nor parity bit is logically inverted.

Inverse convention: ba is msb of data byte to be sent. bh is lsb.
Both the data bits and parity bit are logically inverted by SIM card.

Figure 26-38. Inverse Convention Versus Direct Convention

26.5.3.8 Receive FIFO

The receive FIFO is implemented inside a sub-block of the receiver. The FIFO depth is 285 bytes and is shared by both SIM module ports. The receive FIFO is accessed through the SIM_RBUF*n* registers.

The receive FIFO is loaded from the receive shift register after the final bit of the current SIM card transmission has been received. The FIFO contains ten bits per transmission. The lower eight bits contain the received data byte. Bits 8 and 9 contain the parity and framing status for the received byte.

Each read from the receive FIFO increments the receive FIFO read pointer. Each time the receive shift register is transferred to the receive FIFO, the receive FIFO write pointer increments. When the difference between the read and write pointers equals the programmed threshold value (RDT), the receive data register full flag (RDRF) is set. An interrupt is generated by RDRF if SIM_IMR[RIM] is cleared. Software has no visibility of the receive FIFO pointers. A write to the receive FIFO register (SIM_RBUF*n*) generates an invalid access exception to the processor.

The receive FIFO is flushed by setting SIM_RCR[RCV_FLUSH]. The flush operation resets the receive read and write pointers to equal values. All logic associated with the receiver will be reset by the flush operation except for receiver control registers.

26.5.3.9 Overrun Detection

The receive FIFO logic is responsible for detecting an overrun condition. When a received byte is transferred from the receive shift register to a receive FIFO that contains 285 unread bytes, the SIM receiver flags an overrun condition (SIM_RSR[OEF]). The received byte is discarded leaving the 285 unread bytes in the FIFO unaltered. The SIM module generates a NACK to the SIM card if SIM_CR[ONACK] is set. The SIM module continually NACK SIM card transmissions until a read of the receive FIFO occurs.

26.5.3.10 Character Wait Time Counter

The SIM receiver block includes a 16-bit counter that counts the number of bit times (ETUs) between received characters. When enabled, the character wait time counter (CWT) does not start counting until after the start bit(s) of a valid character are received. The counter is synchronized to the receive character bit positions so that an accurate count of the number of ETUs between characters is made. The CWT has a 16-bit programmable comparator that software can write the expected number of ETUs between characters to. If the time between characters exceeds this value, an interrupt flag is set and an interrupt generated if the mask is clear.

26.5.4 SIM Port Control

The port control block comprises the following functions: SIM card interface, SIM Card presence detect, and SIM card auto-power down.

26.5.4.1 SIM Card Interface

The SIM module allows for direct control of two separate SIM cards. The SIM module does not support simultaneous communication with two SIM cards.

The SIM card clock is generated in the SIM clock generator. The SIM card reset and card voltage enable are controlled by software through SIM_CR*n*.

See Figure 26-39 for an example SIM module connection to two SIM cards. The power management chip shown is used to provide Vcc for the SIM card, and level translators for the remaining signals when interfacing to a SIM card operating at a different voltage than SIM module. The SIM module can directly access a SIM card if it is operating at the same voltage.

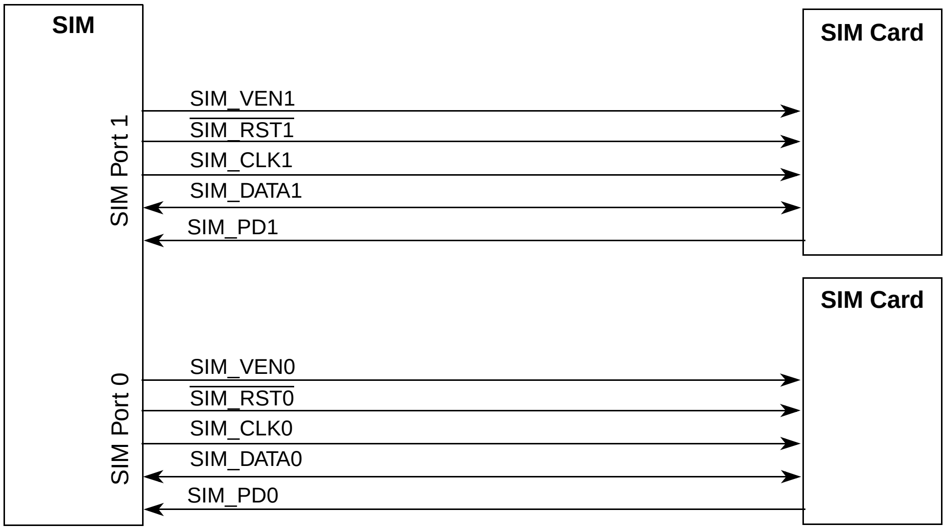


Figure 26-39. SIM Card Hookup

26.5.4.2 SIM Card Presence Detect

The SIM_PD*x* input allows for detection of the insertion or removal of a SIM card. Software can use SIM_DETECT*n*[SPDS] to configure which edge of the SIM_PD*x* pin causes a presence detect event. A maskable interrupt can be generated when a SIM_PD*x* event occurs.

You can place an external pull-down resistor on the SIM_PD*x* pins. Doing so allows a high-to-low transition on the SIM_PD*x* pins to occur, when an external driver drives a logic high on the SIM_PD*x* pins when a card is present and the SIM card is then removed.

26.5.4.3 SIM Card Automatic Power Down

When interfacing to the SIM cards, it is necessary to follow a particular sequence when powering them up and down. The SIM port control block contains hardware that provides the correct sequence to power down a SIM card (see Figure 26-40). The power-up sequence must be done manually by the software using the pin control bits supplied in SIM_CR*n*.

The power-down sequence is specified in ISO 7816 as:

1. SIM_RST transitions from high to low.
2. SIM_CLK is negated.
3. I/O transitions from high impedance to low.
4. SIM Vcc is turned off.

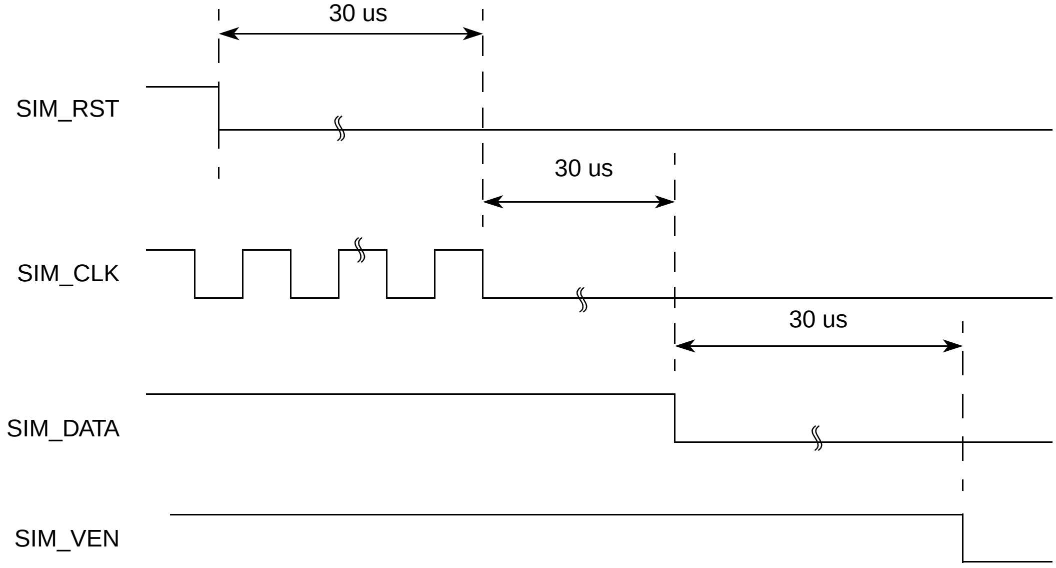


Figure 26-40. Auto Power Down Sequence

26.5.5 SIM General Purpose Counter

The SIM module provides a 16-bit counter for timing events during SIM card communication. The clock source for the counter is selectable between three sources: BAUD_CLK, RCV_CLK, or XMT_CLK (ETU clock). SIM_CR[GPCNT_CLKSEL] selects the clock input. The counter is enabled as soon as the input clock is selected. The starting of the counter is immediate once the input clock is running. Software controls the three input clock sources by using SIM_RCR[KILL_CLOCK] and SIM_EN[RXEN, TXEN].

Table 26-30. General Purpose Counter Clock Source Selections

	SIM_RCR [KILL_CLK]	SIM_CR [GPCNT_ CLKSEL]	SIM_EN [RXEN]	SIM_CR [CWTEN]	SIM_EN [TXEN]
Card Clock Source	0	01	—	—	—
Receive Clock Source	0	10	1	—	—
	0	10	—	—	1
ETU (Transmit) Clock Source	0	11	1	1	—
	0	11	—	—	1

The counter is reset by clearing SIM_CR[GPCNT_CLKSEL]. A 16-bit comparator value allows software to select a count value to interrupt the processor if the mask is clear.

26.5.6 SIM LRC Block

The SIM module provides an 8-bit linear redundancy check (LRC) generator/checker for T=1 SIM cards that support LRC. This block is enabled through SIM_CR[LRCEN] and performs an 8-bit exclusive-OR on all received or transmitted characters. At the end of the reception of a block of characters, the result is expected to be zero. If so, SIM_RSR[LRCOK] is set. During transmission, the LRC block exclusive-ORs each character that is transmitted with the current value of the LRC. If SIM_CR[XMT_CRC_LRC] is set, the LRC value is automatically sent by the SIM transmitter as the final character when the transmit FIFO empties.

The LRC value is reset by any of the following:

- Clearing SIM_CR[LCREN]
- At the end of a transmission (either after the LRC byte is transmitted, or after the last character in the transmit FIFO is sent when XMT_CRC_LRC is clear)
- Setting SIM_EN[TXEN]

26.5.7 SIM CRC Block

The SIM module provides an 16-bit cyclic redundancy check (CRC) generator/checker for T=1 SIM cards that support CRC. This block is enabled through SIM_CR[CRCEN]. This block performs a polynomial based check on all received or transmitted characters. The polynomial description is shown in Figure 4-17. The polynomial used is the standard CRC-CCITT where $g(x) = x^{16} + x^{12} + x^5 + 1$. The CRC register is initialized to all ones before the data is shifted in. Before transmission the resulting CRC is inverted. For example, transmitting a 0xFA results in the following:

- Data = 0x5F (bit reversal of 0xFA)
- CRC = 0x4AEA
- Invert the crc = 0xB515 (bit reversal of 0xADA8)
- Data transmitted = 0xFA, 0xAD, 0xA8

See Figure 26-41 for an illustration of the SIM CRC block.

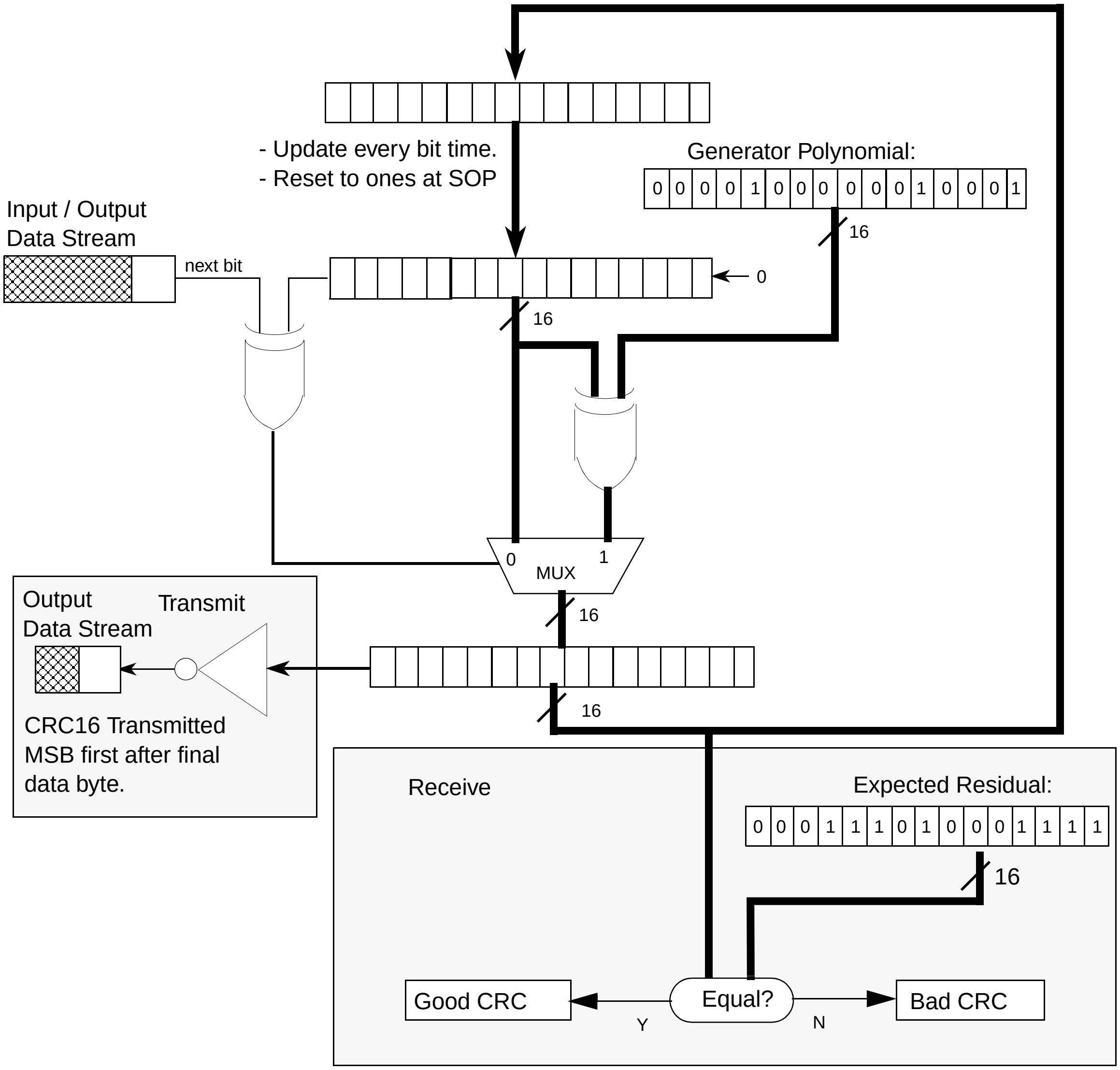


Figure 26-41. CRC Block Diagram

At the end of the reception of a block of characters, the residual from the CRC calculation is compared to 0x1D0F. If equal, SIM_RSR[CRCOK] is set. During transmission, the CRC block updates the current value of the CRC residual using each character. If SIM_CR[XMT_CRC_LRC] is set, the CRC value is automatically inverted and sent by the SIM transmitter as the final two characters when the transmit FIFO empties.

The CRC value is reset by any of the following:

- Clearing SIM_CR[CRCEN]
- At the end of a transmission (either after the CRC characters are transmitted, or after the last character in the transmit FIFO is sent when XMT_CRC_LRC is cleared)
- Setting SIM_EN[TXEN]

26.5.8 Module Interrupts

See Table 26-31 for the list of all possible interrupt sources and their corresponding mask bits. The SIM interrupts are logically ORed to create two interrupts to the interrupt controller:

- SIM data interrupt — includes five transmit and receive status interrupts
- SIM general interrupt — includes ten various SIM interrupts

All mask bits disable the interrupt when set, whereas a zero implies that the interrupt is enabled (unmasked). All mask bits are set out of reset (all interrupts are masked).

Table 26-31. SIM Module Interrupts

Flag	Mask Register	Description
Data Interrupt		
SIM_TSR[TC]	SIM_IMR[TCIM]	Transmit complete
SIM_TSR[ETC]	SIM_IMR[ETCIM]	Early transmit complete
SIM_TSR[TFE]	SIM_IMR[TFEIM]	Transmit FIFO empty
SIM_TSR[TDTF]	SIM_IMR[TDTFM]	Transmit data threshold flag
SIM_RSR[RDRF]	SIM_IMR[RIM]	Receive data register full (FIFO threshold level reached)
General Interrupt		
SIM_TSR[GPCNT]	SIM_IMR[GPCNTM]	General purpose counter comparator flag
SIM_TSR[XTE]	SIM_IMR[XTM]	Transmit threshold error
SIM_TSR[TFO]	SIM_IMR[TFOM]	Transmit FIFO overfill error
SIM_RSR[OEF]	SIM_IMR[OIM]	Overrun error flag
SIM_RSR[CWT]	SIM_IMR[CWTM]	Character wait time counter comparator flag
SIM_RSR[BWT]	SIM_IMR[BWTM]	Block wait time counter comparator flag
SIM_RSR[BGT]	SIM_IMR[BGTM]	Block guard time counter comparator flag
SIM_RSR[RTE]	SIM_IMR[RTM]	Receive NACK threshold error
SIM_DETECT1[SDI1]	SIM_DETECT1[SDIM1]	SIM detect interrupt for port 1
SIM_DETECT0[SDI0]	SIM_DETECT0[SDIM0]	SIM detect interrupt for port 0

26.6 Initialization/Application Information

This section describes the intended programming model for using the SIM module. The section describes how to begin a typical mode of operation using the registers.

26.6.1 Configuring SIM for Operation

The following list of items must be performed to configure the SIM module for operation:

1. Port selection in the SIM_SETUP register.
   a) Select which SIM module port is active by writing the SPS bit.
2. Enable the selected port (port 1, SIM_CR1; port 0, SIM_CR0) following the SIM power-up procedure specified in ISO 7816.
   a) Set SVEN to enable power to the SIM card.
   b) Set STEN to enable SIM module transmit data output. This is required to allow the SIM receiver to create NACK pulses.
   c) Set SCEN to enable SIM card clock.
   d) Set SRST to release the SIM card from reset.
3. Choose the port baud rate in SIM_CR.
   a) Select the SIM card clock rate by using CLK_PRE.
   b) Select the SIM card baud rate by using BAUD_SEL and SAMPLE12

NOTE

Follow the ISO 7816 spec with regards to card clock frequencies to ensure the maximum frequency specification is not violated.

4. Select data format type, or place SIM receiver in initial character mode.
   a) Select inverse convention or direct convention by using SIM_FORMAT[IC], or
   b) Enable initial character mode by using SIM_CR[ICM]

26.6.1.1 Configuring SIM Receive

The following list of items must be performed to configure the SIM receiver for operation:

1. Enable NACK capability in SIM_CR.
   a) Select NACK generation on parity errors, or invalid initial character by using ANACK.
   b) Select NACK generation on overrun conditions by using ONACK.
2. Select the desired receive NACK threshold and receive FIFO threshold in SIM_RTHR.
   a) Program the threshold at which the RDRF flag is set by using RDT.
   b) Program the threshold at which the RTE flag is set by writing to RTH. If an automatic power down after RTE flag is set is desired, then enable the SIM card auto power down by setting SIM_CR*n*[SAPD].
3. Configure the character wait time counter, the block wait timer counter, and the block guard time counter.
4. Enable interrupts in SIM_IMR.
   a) Enable the receive data register full interrupt by using RIM.
   b) Enable the receive threshold error interrupt by using RTM.
   c) Enable the overrun condition interrupt by using OIM.

d) Enable the character wait time interrupt by using CWTM.

26.6.1.2 Configuring SIM Transmitter

The following list of items must be performed to configure the SIM transmitter for operation:

1. Select desired re-transmission threshold for NACKed characters in SIM_TTHR.
   a) Program the threshold at which the XTE flag will be set by using XTH.
2. Select the guard time between transmissions in SIM_TGCR.
   a. Program the desired guard time between characters transmitted by the SIM module by using GETU.
3. Select the desired transmit FIFO threshold level in SIM_TTHR.
   a) Program the desired threshold using TDT.
4. Enable interrupts in SIM_IMR.
   a) Enable the transmit complete interrupt by using TCIM.
   b) Enable the early transmit complete interrupt by using ETCIM.
   c) Enable the transmit FIFO empty interrupt by using TFEIM.
   d) Enable the transmit threshold error interrupt by using XTM.
   e) Enable the transmit FIFO threshold interrupt by using TDTFM.
   f) Enable the transmit FIFO overfill interrupt by using TFOM.

26.6.1.3 Configuring SIM General Purpose Counter

The following list of items must be performed to configure the SIM general purpose counter for operation:

1. Select the desired clock source for the general purpose counter using the SIM_CR register.
   a) Use SIM_CR[GPCNT_CLKSEL] to select the desired clock source for the counter
2. Program counter comparator using the SIM_GPCNT register.
   a) Use the GPCNT bits to select the desired count value at which the GPCNT interrupt flag is set.
3. Enable the selected clock source for the general purpose counter using SIM_RCR or SIM_EN.
   a) If the GP counter is configured for the card clock, enable the clock by clearing SIM_RCR[KILL_CLOCK].
   b) If the GP counter is configured for the receive oversample clock, enable this clock by setting SIM_EN[RXEN] or SIM_EN[TXEN].
   c) If the GP counter is configured for the transmit oversample clock, enable this clock by setting SIM_EN[TXEN].
4. Enable interrupts in SIM_IMR.
   a) Enable the general purpose counter interrupt by clearing GPCNTM.

26.6.1.4 Configuring SIM to Measure WWT (Work Wait Time) for Type=0 Smart Cards

Work wait time is a combination of BWT and CWT. If you want the SIM to enforce a WWT of 100 bits, then you must activate CWT and BWT, and program them both for a value of 100 bits. When measuring WWT, the BGT timer is not used so it is masked (inactive) by the BGTM bit.

Below is the sequence to program the SIM to send and receive data while checking WWT.

1. Program SIM_CWTR, SIM_BWTL, and SIM_BWTH registers to the WWT (work wait time) value that needs to be enforced. Clear the BGT register.
2. Activate both the CWT and BWT functions by setting SIM_CR[CWTEN, BWTEN].
3. Enable the CWT and BWT interrupts by clearing SIM_IMR[CWTM, BWTM].
4. Program the data to send to the smart card by writing data to SIM_TBUF*n*.
5. Activate the SIM transmit and receive by setting SIM_EN[TXEN, RXEN].
6. If the software has more data to send, then as the FIFO starts to empty, it should write more data to SIM_TBUF*n*. If the software does not have any more data to send then proceed to the next step.
7. When the smart card has completed its transmission to the SIM module, disable the BWT by clearing SIM_CR[BWTEN]. This prepares the SIM to measure the next block wait time.
8. Disable the SIM transmitter by clearing SIM_EN[TXEN].
9. Program the next data to send by writing data to SIM_TBUF*n*.
10. Enable the BWT function by setting SIM_CR[BWTEN].
11. Enable the SIM transmitter by setting SIM_EN[TXEN].
12. Steps 6 to 10 can be repeated for each transmission to the smart card.
13. If a CWT or a BWT interrupt occurs, the software should consider this a WWT violation. The software should clear the CWT or BWT interrupt by writing one to SIM_RSR[BWT or CWT].

26.6.1.5 Configuring SIM to measure CWT, BWT, BGT for Type=1 Smart Cards

The following list of items must be performed to configure the SIM to measure CWT, BWT, and BGT.

1. Program SIM_CWTR, SIM_BWTL, SIM_BWTH, and SIM_BGT to the enforced value.
2. Activate the CWT and BWT functions by setting SIM_CR[CWTEN, BWTEN].
3. Enable the CWT, BWT, and BGT interrupts by clearing SIM_IMR[CWTM, BWTM, BGTM].
4. Program the data to send to the smart card by writing data to SIM_TBUF*n*
5. Activate the SIM transmit and receive by setting SIM_EN[TXEN, RXEN].
6. As the FIFO approaches empty, if the software has more data to send, it should write more data to SIM_TBUF*n*. If the software does not have any more data to send, proceed to the next step.
7. When the smart card has completed its transmission to the SIM module, disable the BWT by clearing SIM_CR[BWTEN. This prepares the SIM to measure the next block wait time.
8. Disable the SIM transmitter by clearing SIM_EN[TXEN].
9. Program the next data to send by writing data to SIM_TBUF*n*.
10. Enable the BWT function by setting SIM_CR[BWTEN].

11. Enable the SIM transmitter by setting SIM[TXEN].
12. Steps 6 to 10 can be repeated for each transmission to the smart card.
13. If a CWT or a BWT interrupt occurs, the software should read SIM_RSR to determine if the error was caused by a CWT, BWT, or BGT violation. The software should clear the CWT, BWT, or BGT interrupt by writing a one to SIM_RSR[CWT, BWT, or BGT]

26.6.1.6 Configuring SIM Linear Redundancy Check (LRC) Block

The following list must be performed to configure the SIM linear redundancy check block for operation:

1. Enable the LRC block by using SIM_CR
   a) Use LRCEN to enable the LRC block
   b) Use XMT_CRC_LRC to enable the transmission of the LRC character after the last character in the transmit FIFO is sent. Refer to the T=1 programming model for more details.

26.6.1.7 Configuring SIM Cyclic Redundancy Check (CRC) Block

The following list must be performed to configure the SIM cyclic redundancy check block for operation:

1. Enable the CRC block by using SIM_CR.
   a) Use CRCEN to enable the CRC block
   b) Use XMT_CRC_LRC to enable the transmission of the CRC characters after the last character in the transmit FIFO is sent. Refer to the T=1 programming model for more details.

26.6.2 Using SIM Receiver

When the SIM has been properly configured (correct baud rate and data format), SIM receptions can be enabled by setting the receive enable bit, SIM_EN[RXEN]. As bytes are received, they are placed in the 285 byte deep receive data FIFO. Unread bytes can be accessed from this FIFO at any time. There is no need to disable the receiver to access the FIFO. The FIFO should only be read when the receive FIFO data flag, SIM_RSR[RFD] is set. The RFD flag, which cannot generate an interrupt, is set when there is at least one unread byte in the receive FIFO. If the receive FIFO is read when RFD is cleared, it will simply produce the last byte read.

The correct address to read the data contained in the receive FIFO depends on which SIM port is selected. SIM_SETUP[SPS] controls port selection. If cleared, read the data from the SIM_RBUF0; if set, read the data from SIM_RBUF1.

The receive data register full flag, SIM_RSR[RDRF], is used to determine when the receive FIFO has reached a given threshold value. This flag generates an interrupt if SIM_IMR[RIM] is cleared. To control at when RDRF is set, program the receive data threshold, SIM_RTHR[RDT]. If the number of unread bytes in the receive FIFO is equal to or greater than the value set by RDT, RDRF is set.

NOTE

A value of 0x0 in RDT implies that there must be 285 unread bytes in the receive FIFO to trigger RDRF.

The value in RDT can be changed at any time to alter this threshold level. The comparison between the number of unread bytes in the FIFO and the value set by RDT is continuously updated so that any change in either will be immediately reflected in the state of RDRF. For instance, if RDT is set to 0x5 and there are three unread bytes in the FIFO, changing RDT to 0x2 immediately sets RDRF. Likewise, setting RDT back to 0x5 clears RDRF. Similarly, if there are five unread bytes in the receive FIFO and RDT is set to 0x3, RDRF remains set until three reads are complete (assuming RDT is constant and no new data is received).

The standard flow for receiving bytes from the SIM card is to:

1. Set RDT to the appropriate value.
2. Wait for RDRF to cause an interrupt (RIM clear).
3. Read bytes out of the receive FIFO as long as RFD is set.

   In addition to checking RFD between every byte, it is also recommended to check for the existence of a set OEF flag as well.

26.6.2.1 Receive Parity Errors and Parity NACK Generation

The SIM receiver checks every byte received for proper parity. SIM_FORMAT[IC] controls whether it checks for odd or even parity. When checking for odd parity, the number of logic ones contained in the nine received bits (eight data bits and one parity bit) should be odd. Likewise, when checking for even parity, the number of logic ones contained in the nine received bits should be even.

When a parity error is detected on a given byte, SIM_RBUF*n*[PE] for that byte is set. The PE flag for each byte is read out of the FIFO when the data itself is read. There is no need to clear the parity error flag in the FIFO. It is simply overwritten the next time a byte is received into that position of the FIFO. A parity error cannot cause an interrupt.

If SIM_CR[ANACK] is set, the SIM automatically requests the SIM card to resend a byte found with a parity error by generating a NACK pulse on the SIM_DATA pin. Bytes with parity errors that cause a NACK pulse are still placed into the FIFO just as bytes that do not cause a NACK pulse. Software must discard data bytes with parity errors.

To control NACK generation by the SIM receiver use SIM_RTHR[RTH]. This set of bits specify the number of consecutive NACKs generated by the SIM module on a received byte, before setting the receive threshold interrupt flag, SIM_RSR[RTE]. The RTE flag also forces the SIM port to power-down the card if SIM_CR*n*[SAPD] is set.

NOTE

> SIM_CR[ANACK] must be set to enable this feature. In initial character mode, ANACK enables the retransmission of initial characters in the event that an invalid initial character is received.

When a valid character has been received by the SIM, the internal counter keeping track of the number of NACKs transmitted on the current byte resets to zero. Clearing SIM_RSR[RTE] also clears that counter.

When generating a NACK pulse, the SIM generates the low pulse starting at 10.5 ETUs and lasting for one ETU.

26.6.2.2 Receive Frame Errors

The SIM receiver checks every byte received for a proper stop bit. A stop bit should exist during at least the first half of the 11th ETU time after the start of the character. If this is not true, a frame error is flagged. When a frame error is detected on a given byte, SIM_TBUF*n*[FE] for that byte is set in the FIFO. The FE flag for each byte is read out of the FIFO when the data itself is read. There is no need to clear the frame error flag in the FIFO. It is simply overwritten the next time a byte is received into that position of the FIFO. A frame error cannot cause an interrupt, nor can it create a NACK pulse to the receiver asking for a retransmission of the corrupted data.

26.6.2.3 Receive Overrun Errors and Overrun NACK Generation

When 285 unread bytes are in the FIFO, a received character will cause the SIM receiver to flag an overrun condition. This condition always sets the overrun error flag, SIM_RSR[OEF]. The received byte is discarded, leaving the 285 unread bytes in the FIFO unaltered.

If SIM_CR[ONACK] is set, the SIM automatically requests the SIM card to resend the byte that caused the overrun condition by generating a NACK pulse on the SIM_DATA pin. In this case, the existence of an OEF flag does not indicate the loss of data, but rather a NACK (retransmission request) due to a full receive FIFO. As opposed to transmit NACK generation, there is no limit to the number of times an overrun condition causes a NACK other than to disable ONACK itself. When generating a NACK pulse, the SIM generates the low pulse starting at 10.5 ETUs and lasting for one ETU.

If ONACK is cleared, a set SIM_RSR[OEF] indicates the loss of data. The OEF flag generates an interrupt if SIM_IMR[OIM] is cleared.

To clear OEF, software must write a one to SIM_RSR[OEF]. A high OEF flag has no effect on the operation of the SIM receiver other than to create an interrupt if OIM is clear.

26.6.2.4 Initial Character Mode and Resulting Receive Data Formats

The SIM receiver supports the detection of special characters that allow it to determine what data format is being used by the connected SIM card. When placed in initial character mode, the SIM expects to receive one of two potential values that it uses to set the data format control bit, SIM_FORMAT[IC].

The two possible data formats are inverse convention and direct convention. Essentially, inverse convention differs from direct convention in that the order of the data is flipped, and the data and parity bit are logically inverted. When receiving inverse convention data, the transformation of the data back to direct convention format is done in hardware, including the inversion of the data and parity bits.

To place the SIM into initial character mode, set SIM_CR[ICM]. When a valid initial character is received, SIM_FORMAT[IC] is set accordingly by the hardware, and ICM is cleared. Software can read the state of IC to determine which mode the SIM is currently using.

A 0x3B (as decoded by direct convention) with parity bit set causes direct convention to be used (IC cleared); whereas, a 0x3F (as decoded by inverse convention) with parity bit set causes inverse convention to be used (IC set).

When the receiver is in initial character mode, all received bytes continue to be placed into the receive FIFO whether they are valid initial characters or not. If a valid initial character is received that causes the data format being used to change, all subsequent bytes are decoded with that format before being placed into the FIFO (including the initial character byte itself). That is, if IC is cleared and the correct initial character for setting inverse convention is received, that character and all subsequently received characters are stored in the FIFO after having been decoded using inverse convention (for example, the initial character is stored as 0x3F).

If the receiver is in initial character mode (ICM is set) and an invalid initial character is received, the SIM can be configured to automatically request that the initial character be retransmitted by setting SIM_CR[ANACK]. When generating a NACK pulse, the SIM generates the low pulse starting at 10.5 ETUs and lasting for one ETU. The invalid initial character is placed into the receive FIFO and marked with a parity error, signifying that this is an invalid initial character.

26.6.2.5 Initial Character Mode Programming Notes

The usage of initial character mode requires close attention to the programming model. A parity error in the initial byte for direct convention (0x3B) could be decoded as what appears as a valid initial character for inverse convention (0x3F). The SIM module does not recognize this as a valid initial character for inverse convention and marks the character by setting the parity error flag. The software must look for the existence of a parity error before recognizing a character as a valid initial character.

26.6.2.6 Automatic Receiver Mode

The SIM module has an automatic receive mode that inhibits the data being transmitted by the SIM module from entering the SIM receive buffer through the feedback path of the SIM data pin. The SIM module receiver should normally be enabled while the transmitter is operational. Automatic receive mode saves the software from having to actively manage the transition from transmitter to receiver. The auto-receive mode is always active when the receiver is enabled.

26.6.2.7 Using the SIM Receiver with T=1 SIM Cards

T=1 SIM cards present several requirements beyond standard T=0 cards. The features provided to meet the requirements that pertain to the SIM receiver are as follows:

- 11 ETU characters
  - T=1 cards can transmit with character lengths of 11 ETUs (one stop bit). The SIM module provides SIM_TGCR[RCVR11] register to configure the receiver state machine to accept 11 ETU characters.
- Character wait time counter
  - The character waiting time (CWT) is the time between the start bits of two consecutive characters received from the smart card. The value of CWT can range from 12 ETU to 32,779 ETU. The SIM module provides a 16-bit counter with programmable comparator clocked at the ETU bit rate to identify when the CWT has been exceeded by the SIM card.
- Block waiting time
  - The block waiting time (BWT) is the maximum time between the start bits of the last character of a transmitted block and the first character of the next received block. The BWT must not

exceed a value that is programmable. The SIM module provides a 32-bit block wait timer (divided in two registers) that identifies when the BWT has been violated.

- Block guard time
  - — The block guard time (BGT) is the minimum delay between the start bits of the last character of a transmitted block and the first character of the next received block. The BGT must be greater than a value that is programmable. The SIM module provides a 16-bit block guard timer that identifies when the BGT has been violated.
- Error detection code
  - — T=1 cards can specify LRC or CRC error detection codes to be used. The SIM module provides hardware support for LRC and CRC operations.

26.6.3 Using SIM Transmitter

Once the SIM is properly configured (data XMT enabled, correct baud rate, and correct data format), the transmitter is enabled by setting SIM_EN[TXEN]. If data was previously written to the transmit FIFO, the transmitter begins to send the first character. If no data is written to the transmit FIFO before enabling the transmitter, then the transmitter waits until the first character is written before beginning transmission. Clearing SIM_EN[TXEN] while the transmitter is in operation, halts any transmission in progress, flushes the transmit FIFO, and resets the transmit state machine. Data can be written to the transmit FIFO at any time.

The transmit data threshold flag, SIM_TSR[TDTF] is used to determine when the transmit FIFO has reached a given threshold value. This flag creates an interrupt if SIM_IMR[TDTFM] is cleared. To control at which point TDTF is set, program the transmit data threshold, SIM_TTHR[TDT]. If the number of bytes remaining in the transmit FIFO is equal to or less than the value set by TDT, TDTF isset.

NOTE

A value of zero in TDT implies that the transmit FIFO must be empty to trigger TDTF.

The value in TDT can be changed at any time to alter this threshold level. The comparison between the number of remaining bytes in the transmit FIFO and the value set by TDT is continuously updated so that any change in either will be immediately reflected in the state of TDTF. Unlike the RDRF flag for the receive FIFO, TDTF is latched and remains set until the software writes a one to SIM_TSR[TDTF]. For instance, if TDT is set to 0x5, and there are six bytes remaining in the FIFO, changing TDT to 0x6 immediately causes TDTF to set. However, setting TDT back to 0x5 does not cause TDTF to clear.

The standard flow for transmitting bytes from the SIM card is to:

1. Set TDT[3:0] to the appropriate value
2. Write up to 16 bytes to the transmit FIFO
3. Enable the transmitter
4. Wait for TDTF to cause an interrupt (TDTFM clear)
5. Write additional bytes to the transmit FIFO

26.6.3.1 Transmit Data Formats

There are two possible data formats the SIM module uses when transferring data to the card: inverse or direct convention. The format used depends on the state of inverse convention bit, SIM_FORMAT[IC]. Software can set the IC bit or it can be set automatically by hardware when using initial character mode.

26.6.3.2 Transmit NACK

The SIM transmitter can respond to NACKs created by the SIM card. A NACK is decoded if the SIM card creates a logic low level on the SIM receive pin during the stop bit time at the end of a transmitted byte. To prevent a situation where the SIM interface is stalled by an infinite number of NACK pulses on a given byte, the SIM module can be configured to limit the number of times it will respond to NACKs by SIM_TTHR[XTH]. If the threshold is reached, a transmit threshold error, SIM_TSR[XTE], is asserted.

When XTE is set, the SIM transmitter is halted, all pending transfers are aborted, and the TC, ETC, and TFE flags are set. All bytes remaining in the transmit FIFO are lost. There is no way to restart the transmission on the next byte in the FIFO. The transmitter remains frozen until XTE is cleared by software by writing a one to SIM_TTHR[XTE]. The XTE flag can generate an interrupt if SIM_TTHR[XTM] is cleared.

You can disable the detection of NACKs from the SIM card by setting XTH to 0x0. Setting XTH to 0x1, disables all retransmissions while still setting XTE on the first NACK received. In general, XTE is set on the NACK that causes the threshold to be reached. This final NACK does not cause a retransmission, whereas all previous NACKs do.

26.6.3.3 Transmit Guard Time

The time between data bytes sent from the SIM transmitter can be altered using the transmit guard time control. By default, the minimum time between start bits of successive transmitted bytes is 12 ETUs (a start bit, eight data bits, a parity bit, and two stop bits). The number of stop bits (idle bits) can be extended by an integer number of ETUs. The number of additional ETUs can be programmed directly into SIM_TGCR[GETU]. Setting GETU bits to 0xFF configures the SIM transmitter to use only one stop bit for each character transmission.

26.6.3.4 Using the SIM Transmitter with T=1 SIM Cards

T=1 SIM cards present several requirements beyond standard T=0 cards. The features provided to meet the requirements that pertain to the SIM transmitter are as follows:

- 11 ETU characters
  - The SIM module transmitter has a programmable guard time register that allows the programmer to specify the number of ETUs between character transmissions. Programming a value of 255 (0xFF) in SIM_TGCR[GETU] sets the number of ETUs per character transmitted to 11.
- Character waiting time
  - The character waiting time (CWT) is the time between the start bits of two consecutive characters. The value of CWT can range from 12–32,779 ETU. The time between transmitted

characters is controlled by the programmable guard time in SIM_TGCR. However, the time between the last byte in the transmit FIFO, and the next transmitted byte can be largely affected by software response time to the transmit interrupts. The SIM transmitter provides a transmit FIFO threshold (TDTF) interrupt to signal the system when the expected number of characters have been transmitted from the transmit FIFO. The minimum CWT is achieved only if the software can respond to the TDTF interrupt and write new data to the transmit FIFO before the last character in the transmit FIFO has been sent.

- Block waiting time
  - The block waiting time (BWT) is the maximum time between the start bits of the last character of a transmitted block and the first character of the next received block. The value of BWT is always greater than 1800 ETU. The SIM transmitter provides a general purpose counter that can be used to track the BWT. The BWT is purely determined by software response time to the transmit interrupts.
- Block guard time
  - The block guard time (BGT) is the minimum delay between the start bits of the last character of a transmitted block and the first character of the next received block. The value of BGT is 22 ETU. The SIM module supports the BGT by providing an interrupt when the last byte is received, and transmitting within 2 ETU after SIM_EN[TXEN] is set. The BGT is determined by the speed at which the software can react to an interrupt and enable the transmitter.
- Error detection code
  - T=1 cards can specify LRC or CRC error detection codes to be used. The SIM module provides hardware support for both LRC and CRC operation.

26.6.4 Suggested Programming Model

This section describes the suggested programming model for supporting T=1, T=0, and known special cards using the SIM module. This should be used as a rough guide for configuring the SIM module for SIM cards specified by ISO 7816-3 and EMV. Some details are not addressed. Other uses for some of the SIM features are not included (for example, GP counter for some ISO timing requirements).

26.6.4.1 Detecting Answer To Reset (ATR)

The first step to communicating with a SIM card is providing power and a clock signal to the card. Once the card is detected as present (using the presence detect features or some other method), the SIM card should be powered up according to the power-up sequence specified in the ISO 7816-3 specification.

1. Apply voltage to the SIM card by setting SIM_CR*n*[SVEN].
2. Select the appropriate clock frequency for the SIM card by programming SIM_PRE.
3. Enable the clock to the SIM card by setting SIM_CR*n*[SCEN].
4. Remove the card from reset by setting SIM_CR*n*[SRST].

   The first communication between the SIM card and the SIM module is a block of data sent from the SIM card to the SIM module after the card is powered and the card reset is removed. This block is called the answer to reset (ATR). To receive the ATR, the SIM module should be configured for

12 ETU character reception. According to the ISO 7816-3 spec, both T=0 and T=1 cards communicate initially using 12 ETU character durations.

NOTE

> We are aware of some card manufacturers that communicate at 11.5 ETU character durations (Geldkarte). This complicates the initial card detection sequence shown below.

5. Clear SIM_TGCR[RCVR11].
6. Set SIM_CR[ANACK] to enable NACK generation.

   The ISO 7816-3 spec allows the SIM module to NACK any communication errors that occur during the initial communication at 12 ETU.

NOTE

> The Europay Mastercard and VISA (EMV) cards are similar to T=1, but do not allow the SIM module to NACK during the initial communication. This will again complicate the initial card detection sequence shown below.

7. Enable RDRF and OEF interrupts by setting SIM_IMR[RIM, OIM] to notify when characters are received.
8. Set desired threshold for received characters before generating an interrupt by writing SIM_RTHR[RDT].
9. Set initial character mode by setting SIM_CR[ICM].

   This causes the hardware to identify the first valid character sent during the ATR as an initial character. This character automatically configures the hardware for the data convention used by the SIM card.

The ISO 7816-3 spec requires that SIM cards meet certain timing restrictions. One of these is the time from the deassertion of the card reset to the beginning of the ATR sequence. The SIM module general purpose counter can verify that the SIM card begins its ATR within the 400 to 40,000 clock cycle range.

10. Set general purpose counter comparator to 0x9C40 using SIM_GPCNT.
11. Enable the general purpose counter interrupt by clearing SIM_IMR[GPCNTM].
12. Enable the general purpose counter by programming SIM_CR[GPCNT_CLKSEL] to 01 so the card clock is used for counting.

The ISO7816-3 spec states that the maximum allowed time between two characters during the ATR is 9600 ETUs (initial waiting time). The character wait time (CWT) counter should be setup to detect any errors for this condition.

13. Set CWT counter comparator to 9600 using SIM_CWTR.
14. Enable the CWT counter interrupt by clearing SIM_IMR[CWTM].
15. Enable the CWT counter by setting SIM_CR[CWTEN].

The last step in preparing for ATR reception is to enable the receiver.

16. Set SIM_EN[RXEN].

The SIM module generates interrupts once a threshold number of characters is received. The software should react to these interrupts and read the characters from the receive FIFO (SIM_RBUF*n*) until the complete ATR is received. If a general purpose counter interrupt occurs before the final ATR character is received, then the card should be deactivated according to ISO 7816-3. Otherwise, once a valid ATR is received, the software knows from the ATR information the specific characteristics for this card (refer to the ISO 7816-3 spec for details).

26.6.4.2 Programming Considerations for Geldkarte Cards

Geldkarte SIM cards do not send the ATR in 12 ETU mode or support NACKs. This creates an issue with detecting a valid ATR. As a result, the software and hardware do not know how to begin communication. Basically, if the card fails to send a valid ATR on the first try, disable NACKs, configure the SIM module for 11 ETU, and try again. The software should toggle between 12 and 11 ETU modes with and without NACKs enabled until a valid ATR is received, or the number of attempts to communicate passes a predetermined error threshold. Figure 26-42 shows the flow chart for the suggested Geldkarte-compliant SIM initialization.

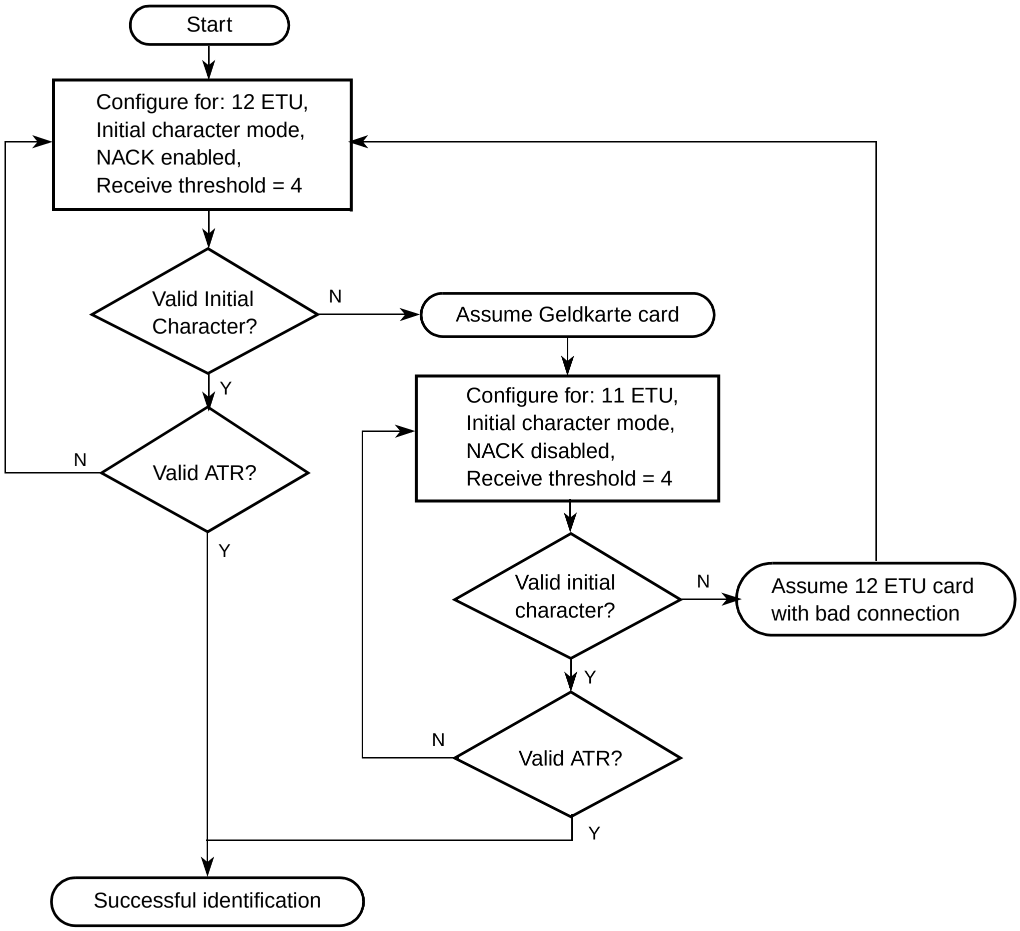


Figure 26-42. Suggested T=1, EMV, Geldkarte Compliant SIM Initialization

26.6.4.3 Programming Considerations for T=0 SIM Cards

If using a T=0 card, software should adjust the following parameters according to the information in the ATR:

1. Adjust the baud rate by changing the values of SIM_CR[BAUD_SEL, SAMPLE12]
2. Adjust the guard time between characters by changing the value of SIM_TGCR[GETU].
3. Adjust NACK capability by modifying the values of SIM_CR[ONACK, ANACK].
4. Adjust the stop clock polarity by modifying the values of SIM_CR*n*[SCSP].
5. Adjust the level of transmit NACK re-transmissions allowed by modifying SIM_TTHR[XTH].
6. Adjust the level for the receive NACK threshold by modifying SIM_RTHR[RTH].

If a negotiation with the SIM card is desired, the software sends a PPS response to the SIM card. To send the response, the following steps must be performed:

1. Set the desired transmit FIFO threshold level by writing SIM_TTHR[TDT].
2. Write the characters to be sent as response (max 16) to the transmit FIFO using SIM_TBUF*n*.
3. Clear all transmit interrupt flags in SIM_TSR by writing a one to them.
4. Enable the desired transmit interrupts by clearing the mask bits in SIM_IMR. If more than 16 characters are sent, use the TDTF interrupt to signify when to write more characters to the transmit FIFO. This results in the most efficient transfer times to the SIM card.
5. Enable the transmitter by setting SIM_EN[TXEN].

At this point, the SIM module transmits the characters in the transmit FIFO. If more than 16 characters are sent, the transmit threshold interrupt is set when the threshold number of characters are remaining in the FIFO. The software can then write an additional number of characters without interrupting transmission to the SIM card.

Once the transmission is complete, the SIM module must be completely configured for standard operation with the T=0 SIM card. The software can continue to service RDRF interrupts for received characters, and TDTF interrupts for transmitted characters.

26.6.4.4 Programming Considerations for T=1 SIM Cards

If using a T=1 card, software should adjust the following parameters according to the information in the ATR:

1. Adjust the baud rate by changing the values of SIM_CR[BAUD_SEL, SAMPLE12]
2. Adjust the guard time between characters by changing the value of SIM_TGCR[GETU]. Setting GETU to 0xFF configures the SIM transmitter for 11 ETU transmissions.
3. Disable NACK capability by clearing SIM_CR[ONACK, ANACK]. T=1 cards do not allow NACKs.
4. Adjust the stop clock polarity by modifying the values of SIM_CR*n*[SCSP].
5. Set character wait time counter comparator to value specified in the ATR by using SIM_CWTR.
6. Enable the character wait time counter interrupt by clearing SIM_IMR[CWTM].
7. Enable the character wait time counter by setting SIM_CR[CWTEN].

8. Enable CRC or LRC error checking according to the ATR information by setting either SIM_CR[CRCEN or LRCEN]. Never set these bits at the same time.

For T=1 cards, the ATR is sent using a T=0 type of structure (12 ETU, no LRC or CRC). If a negotiation with the SIM card is desired, software must send a PPS response to the SIM card. Otherwise, the protocol is initiated with a block transfer from the SIM module. To send the response or the first block, the following steps must be performed:

1. Set the desired transmit FIFO threshold level by writing to SIM_TTHR[TDT].
2. Write the characters to be sent as response (max 16) to the transmit FIFO using SIM_TBUF*n*
3. Clear all transmit interrupt flags in SIM_TSR by writing a one to them.
4. Enable the transmit interrupts desired by clearing the mask bits in SIM_IMR. If more than 16 character are sent, use the TDTF interrupt to signify when to write more characters to the transmit FIFO. This results in the most efficient transfer times to the SIM card.
5. Enable transmission of the error checking characters (LRC or CRC) by setting SIM_CR[XMT_CRC_LRC].

NOTE

If the card supports PPS, the software may not be allowed to send the LRC/CRC information until the PPS exchange is completed. If so, do not set the XMT_CRC_LRC bit during the PPS exchange.

6. Enable the transmitter by setting SIM_EN[TXEN].

At this point, the SIM module transmits the characters in the transmit FIFO. If more than 16 characters are sent, the transmit threshold interrupt is set when the threshold number of characters are remaining in the FIFO. The software can then write an additional number of characters without interrupting transmission to the SIM card.

Once the transmission is complete, the SIM module must be completely configured for standard operation with the T=1 SIM card. The software can continue to service RDRF interrupts for received characters, and TDTF interrupts for transmitted characters.

Chapter 27
Voice Codec

27.1 Introduction

The voice-band audio codec consists of voice coding and decoding. The voice coding function takes a human speech signal in voltage format and converts it to an 8 kHz digital signal. The voice decoding function takes an 8 kS/s digital signal and converts it to an audible voice signal in voltage format.

There are also four audio amplifiers integrated with the voice codec:

- 1 microphone amplifier
- 3 driver amplifiers
  - Speaker
  - Handset
  - Headphone

Only one of the driver amplifiers should be powered up at a time. The amplifiers can be individually bypassed if necessary.

27.1.1 Overview

The oversampled voice codec A/D path receives voice input from a pair of differential inputs, CODEC_ADCP and CODEC_ADCN. The ADC inputs are driven by the integrated microphone amplifier. The microphone amplifier can be bypassed and an external amplifier can be used, if needed. If the micophone amplifier is bypassed, the ADC inputs are AC-coupled with the processor driving the codec using 100 nF capacitors at the inputs. The common mode DC levels are derived from Vag, the common mode voltage generated in the reference block. This configuration ensures the DC level at the ADC inputs is governed by the reference, while its AC input comes from the processor driving the codec.

The ΣΔ A/D converter’s output is filtered to remove the quantization noise and outband signals components and decimated down to twice the Nyquist rate (16 kHz) by a comb digital filter. In the next stage the signal is gain-aligned, low-pass filtered by an IIR filter, and decimated by two. The fourth-order, low-pass IIR filter compensates for passband droop introduced by the comb filter. The 8 kHz signal is optionally high-pass filtered by a second-order IIR filter and stored in the CODEC_RX register from where it is read by software.

The oversampling voice codec D/A path consists of:

- Optional high-pass second-order IIR filter
- Interpolation by two
- Fourth-order IIR low-pass filter

- Gain alignment
- Interpolating digital comb filter
- ΣΔ D/A converter
- Analog smoothing filter
- Differential output buffer to drive the speakers

The low-pass IIR filter also compensates for passband droop introduced by the comb filter. The comb filter interpolates the signal from twice the Nyquist rate up to the oversampled ΣΔ frequency.

The voice codec design supports a number of input clock frequencies. The A/D and D/A oversampling frequency is determined by the input clock and is equal to a quarter of the internal bus frequency:

$$f_{\Sigma\Delta} = \frac{f_{SYS/3}}{4} \qquad \textit{Eqn. 27-1}$$

27.1.2 Block Diagram

Figure 27-1 shows the top level block diagram of the codec functional partitioning. The ADC and DAC converters are implemented with sigma-delta (ΣΔ) oversampled modulators. The integrated codec amplifiers may be bypassed.

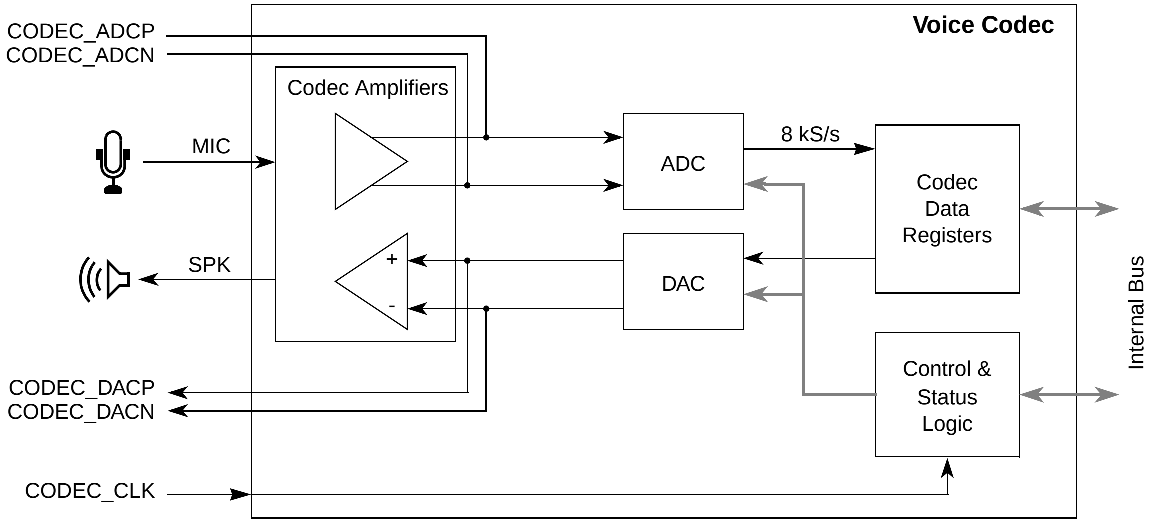


Figure 27-1. Voice Codec Block Diagram

27.2 External Signal Descriptions

Table 27-1. Codec Signal Properties

Name	Function	I/O
Codec Signals		
CODEC_ADCN	Differential analog input to the A/D of the voice codec. By default, these signals are driven by the internal microphone amplifier. If the microphone amplifier is bypassed, they can be driven externally.	I
CODEC_ADCP		I

Table 27-1. Codec Signal Properties (continued)

Name	Function	I/O
CODEC_DACN	Differential analog outputs of the D/A of the voice codec. By default, the DAC outputs drive the speaker, handset, and headphone audio driver amplifiers. If the audio amplifiers are bypassed, the DAC outputs are driven externally and external amplifiers can be used.	O
CODEC_DACP		O
CODEC_ALTCLK	Optional external clock signal to the codec. The voice codec supports seven clock frequencies: 20 MHz, 24 MHz, 26 MHz, 28 MHz, 30 MHz, 19.44 MHz and 16.8 MHz.	I
AVDD_CODEC	Analog supply voltage for the codec regulator and the bandgap reference blocks.	I
CODEC_BGRVREF	Bandgap output voltage. Connect with a 0.1 µF bypass capacitor to VSS_CODEC.	I
CODEC_REGBYP	Output of codec regulator to provide power supply to the ADC, DAC, REF, and microphone amplifier blocks. Connect with a 1 µF external capacitor to VSS_CODEC.	I
CODEC_REFN	Differential reference voltage signals for the analog part of the ADC. Connect each with 0.1 a µF bypass capacitor to VSS_CODEC. **Note:** It is important to connect these capacitors to the same point on the VSS_CODEC trace.	I
CODEC_REFP		I
CODEC_VAG	Analog common mode reference for analog ADC and DAC part of voice codec and audio amplifiers. Connect with a 0.1 µF bypass capacitor to VSS_CODEC.	I
VSS_CODEC	Analog ground.	I
Codec Amplifier Signals		
AMP_HSN	Differential analog output signals of the headset amplifier.	O
AMP_HSP		O
AMP_HPOUT	Single-ended analog output. An onboard AC coupling cap should be connected between AMP_HPOUT and the load.	O
AMP_HPDUMMY	AC coupling of the dummy load of the headphone amplifier to GND. The dummy load is needed for stability of headphone amp for high-capacitance, low-conductance loads. The capacitor to GND at AMP_HPDUMMY output is also used for slow ramp-up of common mode voltage for headphone amplifier.	O
AMP_MICN	Differential analoginput signals of the microphone amplifier.	I
AMP_MICP		I
AMP_SPKRN	Differential analog output signals of the speaker amplifier.	O
AMP_SPKRP		O
AVDD_SPKR	Dedicated power supply pin for the three amplifiers.	I
AVSS_SPKR_HDST	Ground supply pin for the three amplifiers. These pins are internally shorted and must be tied together externally.	I
AVSS_SPKR_HP		I

27.3 Memory Map/Register Definition

The voice codec module contains 12 registers.

Table 27-2. Voice Codec Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC09_C000	Codec control register (CODEC_CR)	16	R/W	0x0000	27.3.1/27-4
0xFC09_C002	Codec status register (CODEC_SR)	16	R	0x4000	27.3.2/27-6
0xFC09_C004	Codec receive register (CODEC_RX)	16	R	0x0000	27.3.3/27-8
0xFC09_C006	Codec transmit register (CODEC_TX)	16	R/W	0x0000	27.3.4/27-8
0xFC09_C008	Codec interpolation rate register (CODEC_IRR)	16	R/W	0x024E	27.3.5/27-8
0xFC09_C00A	Codec gain compensation register (CODEC_GCR)	16	R/W	0x0070	27.3.6/27-9
0xFC09_C00C	Codec RXACQ count register (CODEC_RXACQ)	16	R	0x0000	27.3.7/27-9
0xFC09_C00E	Amplifiers bypass control register (AMPS_BYP)	16	R/W	0x0000	27.3.8/27-10
0xFC09_C010	Amplifiers driver control register (AMPS_DCR)	16	R/W	0x0000	27.3.9/27-11
0xFC09_C012	Amplifiers control register (AMPS_CR)	16	R/W	0x0000	27.3.10/27-12
0xFC09_C014	Amplifiers status register (AMPS_SR)	16	R	0x0000	27.3.11/27-13

27.3.1 Voice Codec Control Register (CODEC_CR)

CODEC_CR determines all codec operation modes. It controls enabling/disabling of the module, enabling module interrupts, determines the operation clock frequency, and activates various test modes.

Address: 0xFC09_C000 (CODEC_CR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	VCEN	VCIE	VC IXE	VC OXE	0	0	BYP DIG	0	VC OHPF	VC IHPF	VC DITH	0	VC MIDI	VCLK		
W												VCRST				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 27-2. Voice Codec Control Register (CODEC_CR)

Table 27-3. CODEC_CR Field Descriptions

Field	Description
15 VCEN	Voice codec enable. 0 Voice codec disabled and placed in power save mode 1 Voice codec enabled
14 VCIE	Voice codec interrupt enable. Indicates that CODEC_TX is empty and a new sample can be output to the DAC. It also indicates that CODEC_RX contains a new input sample from the ADC. 0 Interrupt is disabled 1 Interrupt is enabled

Table 27-3. CODEC_CR Field Descriptions (continued)

Field	Description
13 VCIXE	Voice codec input exception interrupt enable. Enables overrun interrupt (a new sample is placed in CODEC_RX before the previous one is read). 0 Input exception interrupt is disabled 1 Input exception interrupt is enabled
12 VCOXE	Voice codec output exception interrupt enable. Enables an underrun interrupt (a new sample required by the DAC but CODEC_TX is not written yet with new data). 0 Output exception interrupt is disabled 1 Output exception interrupt is enabled
11–10	Reserved, must be cleared.
9 BYPDIG	Bypass digital filters. Dedicated test bit. For normal operation should be written with zero. 0 Normal operation 1 Bypass all digital filters on DAC path
8	Reserved, must be cleared.
7 VCOHPF	Voice codec output high-pass filter enable. Enables the high-pass filter in the DAC path. This bit can be changed on-the-fly even during codec operation. If VCOHPF is changed in the presence of audio, a pop noise may be produced. 0 HPF in DAC path disabled and placed in power-save mode. The signal from the CODEC_TX register bypasses the HPF and enters the LPF. 1 HPF in DAC path enabled
6 VCIHPF	Voice codec input high-pass filter enable. Enables the high-pass filter in the ADC path. The bit can be changed on-the-fly even during codec operation. If VCIHPF is changed in the presence of audio, a pop noise may be produced. 0 HPF in ADC path disabled and placed in power-save mode. The signal from the LPF bypasses the HPF and enters the CODEC_RX register. 1 Voice codec HPF at ADC path enabled
5 VCDITH	Voice codec output dither disable. Dithering decorrelates the periodic modulator quantization noise of the output converter. 0 Output dithering is enabled 1 Output dithering is disabled
4 VCRST	Voice codec reset. Resets the voice codec logic. The bit is self-clearing and reads as zero. 0 No reset 1 Reset voice codec

Table 27-3. CODEC_CR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3
VCMIDI</td><td>Voice codec MIDI mode control. Allows the sampling rate to be programmable. The interpolation rate and gain compensation are programmed through CODEC_IRR and CODEC_GCR. In this mode, the ADC is disabled.
0 Voice mode. Interpolation rate and gain compensation determined from VCLK
1 MIDI mode. Interpolation rate and gain compensation programmed through CODEC_IRR and CODEC_GCR</td></tr>
<tr><td>2–0
VCLK</td><td>Voice codec clock select. Selects the codec clock input and output frequencies. Changing the VCLK bits when VCEN is set causes a reset of the codec logic.
<table>
<tr><th>VCLK</th><th>CODEC_CLK</th><th>Software Interface Rate</th></tr>
<tr><td>000</td><td>16.8 MHz</td><td>8 kHz</td></tr>
<tr><td>001</td><td>19.44 MHz</td><td>8 kHz</td></tr>
<tr><td>010</td><td>19.44 MHz</td><td>8.1 kHz</td></tr>
<tr><td>011</td><td>20 MHz</td><td>8 kHz</td></tr>
<tr><td>100</td><td>24 MHz</td><td>8 kHz</td></tr>
<tr><td>101</td><td>26 MHz</td><td>8 kHz</td></tr>
<tr><td>110</td><td>28 MHz</td><td>8 kHz</td></tr>
<tr><td>111</td><td>30 MHz</td><td>8 kHz</td></tr>
</table></td></tr>
</table>

27.3.2 Voice Codec Status Register (CODEC_SR)

CODEC_SR contains status indicators of the voice codec data registers. It includes indicators of transmit data empty, receive data full, and underrun/overrun situations.

Address: 0xFC09_C002 (CODEC_SR) Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	VCRF	VCTE	VROE	VTUE	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-3. Voice Codec Status Register (CODEC_SR)

Table 27-4. CODEC_SR Field Descriptions

Field	Description
15 VCRF	Voice codec receive data full. Indicates that CODEC_RX contains data from the ADC. This bit is set when data is transferred from the A/D last filter stage output to CODEC_RX. VCRF is auto-cleared when: • CODEC_RX is read • CODEC_CR[VCEN] is cleared • CODEC_CR[VCRST] is set • Hardware reset • Stop instructions If CODEC_CR[VCIE] is set, VCRF assertion generates a voice codec interrupt. 0 CODEC_RX is empty (contains old data) 1 CODEC_RX contains data from the ADC
14 VCTE	Voice codec transmit data empty. Indicates that CODEC_TX is empty and can be written. VCTE sets when: • The data is transferred from CODEC_TX to the DAC filter input • CODEC_CR[VCEN] is cleared • CODEC_CR[VCRST] is set • Hardware reset • Stop instructions VCTE is cleared when CODEC_TX is written. If CODEC_CR[VCIE] is set, VCTE assertion generates a voice codec interrupt. 0 CODEC_TX register is not empty 1 CODEC_TX register is empty and a new data sample can be sent out
13 VROE	Voice codec receive overrun error. Indicates a new sample is received from the codec while the previous received sample in CODEC_RX was not read (overrun error). In this case, the previous received sample is overwritten in CODEC_RX. VROE is cleared by: • Hardware or software reset • Stop instruction • Reading CODEC_SR (with VROE set) followed by reading CODEC_RX • Setting CODEC_CR[VCRST] Clearing CODEC_CR[VCEN] does not affect VROE. If CODEC_CR[VCIXE] is set, VROE assertion generates a codec receive with overrun error interrupt. 0 No overrun errors 1 Receive overrun error **Note:** The overrun interrupt is only an exception indicator. When there is an overrun situation, you cannot count on the data in CODEC_RX to be correct.
12 VTUE	Voice codec transmit underrun error. Indicates a sample was transmitted to the codec section CODEC_TX is empty (underrun error). In this case, the previous received sample is re-transmitted to the DAC. VTUE is cleared by: • Hardware or software reset • Stop instruction • Reading the CODEC_SR with VTUE set followed by writing CODEC_TX • Setting CODEC_CR[VCRST] Clearing CODEC_CR[VCEN] does not affect VTUE. If CODEC_CR[VCOXE] is set, VTUE assertion generates a voice codec transmit with underrun error interrupt. 0 No underrun errors 1 Transmit underrun error **Note:** The underrun interrupt is only an exception indicator. When there is an underrun situation do not count on the data in CODEC_TX.
11–0	Reserved, must be cleared.

27.3.3 Voice Codec Receive Register (CODEC_RX)

The voice codec receive data register contains the last sample converted by the ADC.

Address: 0xFC09_C004 (CODEC_RX) Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RX													0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-4. Voice Codec Receive Data Register (CODEC_RX)

Table 27-5. CODEC_RX Field Descriptions

Field	Description
15–3 RX	Voice codec receive data register. Contains the last sample converted by the ADC. The register is loaded with 13-bit data from the last filter (LPF or HPF) at a rate of 8 kHz (or 8.1 kHz). This transfer operation sets CODEC_SR[VCRF], and if CODEC_CR[VCIE] is set, a VCI interrupt is generated. Reading CODEC_RX clears VCRF.
2–0	Reserved, must be cleared.

27.3.4 Voice Codec Transmit Data Register (CODEC_TX)

The voice codec transmit data register is used for output of voice data samples to the codec DAC.

Address: 0xFC09_C006 (CODEC_TX) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TX															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-5. Voice Codec Transmit Data Register (CODEC_TX)

Table 27-6. CODEC_TX Field Descriptions

Field	Description
15–0 TX	Voice codec transmit data register.Used to write samples to the DAC. The new data should be written after a VCI interrupt (or when CODEC_SR[VCTE] is set). Writing to CODEC_TX clears CODEC_SR[VCTE].

27.3.5 Codec Interpolation Rate Register (CODEC_IRR)

The voice codec interpolation ratio for MIDI mode is stored in CODEC_IRR register.

Address: 0xFC09_C008 (CODEC_MIDIIR) 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	IR										
W																
Reset	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	0

Figure 27-6. Voice Codec Interpolation Rate Register (CODEC_IRR)

Table 27-7. CODEC_IRR Field Descriptions

Field	Description
15–11	Reserved, must be cleared.
10–0 IR	Voice codec MIDI interpolation ratio. Contains the interpolation ratio for MIDI DAC operating mode. This interpolation ratio is selected when CODEC_CR[VCMIDI] is set.

27.3.6 Codec Gain Compensation Register (CODEC_GCR)

The voice codec gain compensation value for MIDI mode is stored in CODEC_GCR.

Address: 0xFC09_C00A (CODEC_GCR) 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	GC							
W																
Reset	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0

Figure 27-7. Voice Codec Gain Compenstation Register (CODEC_GCR)

Table 27-8. CODEC_GCR Description

Field	Description
15–8	Reserved, must be cleared.
7–0 GC	Voice codec MIDI gain compensation. Contains the gain compensation value for MIDI DAC operating mode. This gain compensation is selected when CODEC_CR[VCMIDI] is set.

27.3.7 Codec RXACQ Count Register (CODEC_RXACQ)

The number of cycles between the rising edge of rx_acq and the rising edge of sdfs is stored in the CODEC_RXACQ register.

Address: 0xFC09_C00C (CODEC_RXACQ) Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RXACQ															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-8. Voice Codec RXACQ Count Register (CODEC_RXACQ)

Table 27-9. CODEC_RXACQ Field Descriptions

Field	Description
15–0 RXACQ	Counts the number of cycles between the rising edge of rx_acq and the rising edge of sdfs. The counter starts after detecting a rising edge on rx_acq and stops after the first rising edge on sdfs. The counter is cleared when the register is read.

27.3.8 Amplifiers Bypass Control Register (AMPS_BYP)

This register provides various bypass options to bypass the speaker, handset, headphone, and microphone amplifiers. This allows use of external components in case internal components do not work as expected.

This register is superceded by the IIM's AMPS_BYPASS fuse register detailed in Section 25.4.11.4, "Amplifier Bypass Register (IIM_AMPBR)".

Address: 0xFC09_C00E (AMPS_BYP) 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	0	0	MIC_BYP	HP_BPY	HS_BYP	SPK_BYP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-9. Amplifiers Bypass Control Register (AMPS_BYP)

Table 27-10. AMPS_BYP Field Descriptions

Field	Description
15–4	Reserved, must be cleared.
3 MIC_BYP	Microphone bypass. If the IIM_AMPBR[MIC] fuse is cleared, this bit determines if the microphone amplifier is bypassed. If IIM_AMPBR[MIC] is set, this bit is ignored. 0 Microphone amplifier is enabled 1 Microphone amplifier is bypassed. The ADC is directly driven by CODEC_ADCP and CODEC_ADCN.
2 HP_BYP	Headphone bypass. If the IIM_AMPBR[HP] fuse is cleared, this bit determines if the headphone amplifier is bypassed. If IIM_AMPBR[HP] is set, this bit is ignored. 0 Headphone amplifier is enabled 1 Headphone amplifier is bypassed. AMP_HPOUT and AMP_HPDUMMY are tri-stated.
1 HS_BYP	Handset bypass. If the IIM_AMPBR[HS] fuse is cleared, this bit determines if the handset amplifier is bypassed. If IIM_AMPBR[HS] is set, this bit is ignored. 0 Handset amplifier is enabled 1 Handset amplifier is bypassed. The DAC outputs are available on CODEC_DACP and CODEC_DACN.
0 SPK_BYP	Speaker bypass. If the IIM_AMPBR[SPK] fuse is cleared, this bit determines if the speaker amplifier is bypassed. If IIM_AMPBR[HS] is set, this bit is ignored. 0 Speaker amplifier is enabled 1 Speaker amplifier is bypassed. AMP_SPKRN and AMP_SPKRP are tri-stated.

27.3.9 Amplifiers Driver Control Register (AMPS_DCR)

AMPS_DCR provides various common options to control all three driver amplifiers (speaker, handset, and headphone).

Address: 0xFC09_C010 (AMPS_DCR)

Access: User read/write

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td rowspan="2">DRV_
MUT</td><td rowspan="2" colspan="3">DRV_VOL</td></tr>
<tr><td>W</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 27-10. Amplifiers Driver Control Register (AMPS_DCR)

Table 27-11. AMPS_DCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–4</td><td>Reserved, must be cleared.</td></tr>
<tr><td>3
DRV_MUT</td><td>Driver amplifier mute.
0 Driver amplifiers are not muted
1 Driver amplifiers are muted</td></tr>
<tr><td>2–0
DRV_VOL</td><td>Driver amplifier volume. Determines the volume setting of the driver amplifiers.
<table>
<tr><th>DR_VOL</th><th>Speaker/ Handset Gain (dB)</th><th>Headphone Amplifier Gain (dB)</th></tr>
<tr><td>000</td><td>0</td><td>–6</td></tr>
<tr><td>001</td><td>–46</td><td>–46</td></tr>
<tr><td>010</td><td>–21.6</td><td>–21.6</td></tr>
<tr><td>011</td><td>–6</td><td>–12</td></tr>
<tr><td>100</td><td>0</td><td>–6</td></tr>
<tr><td>101</td><td>4</td><td>–2</td></tr>
<tr><td>110</td><td>6</td><td>0</td></tr>
<tr><td>111</td><td>6</td><td>0</td></tr>
</table></td></tr>
</table>

27.3.10 Amplifiers Control Register (AMPS_CR)

This register serves as a control register for all the amplifiers: microphone, speaker, handset, and headphone amplifiers.

Address: 0xFC09_C012 (AMPS_CR) Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	HP_ ST	HP_ EN	0	HS_ EN	0	SPK EN	0	0	0	MIC MUT	MIC_VOL			MIC EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-11. Amplifiers Control Register (AMPS_CR)

Table 27-12. AMPS_CR Field Descriptions

Fields	Description
15–14	Reserved, must be cleared.
13 HP_ST	Headphone start-up time. Determines the start-up time (HST) for the headphone amplifier. 0 250 ms 1 500 ms
12 HP_EN	Headphone enable. Enable signal for the headphone amp. 0 Headphone amp is powered down 1 Headphone amp is powered up
11	Reserved, must be cleared.
10 HS_EN	Handset enable. Enable signal for the handset amp. 0 Handset amp is powered down 1 Handset amp is powered up
9	Reserved, must be cleared.
8 SPKEN	Speaker enable. Enable signal for the speaker amp. 0 Speaker amp is powered down 1 Speaker amp is powered up
7–5	Reserved, must be cleared.
4 MICMUT	Microphone amplifier mute. 0 Microphone amp is unmuted 1 Microphone amp is muted

Table 27-12. AMPS_CR Field Descriptions (continued)

Fields	Description
3–1 MIC_VOL	Microphone amplifier volume. Determines the gain setting of the microphone amplifiers. <table><tr><th>MIC_VOL</th><th>Gain (dB)</th></tr><tr><td>000</td><td>0</td></tr><tr><td>001</td><td>6</td></tr><tr><td>010</td><td>9.56</td></tr><tr><td>011</td><td>15.56</td></tr><tr><td>100</td><td>20</td></tr><tr><td>101</td><td>24</td></tr><tr><td>110</td><td>29.56</td></tr><tr><td>111</td><td>39.92</td></tr></table>
0 MICEN	Microphone enable. Enable signal for the microphone amp. 0 Microphone amp is powered down 1 Microphone amp is powered up

27.3.11 Amplifiers Status Register (AMPS_SR)

The bits in this register are controlled by the logic embedded with the amplifiers. They indicate if there is a problem with normal amplifier operation. This register serves as a control register for all the amplifiers: microphone, speaker, handset, and headphone amplifiers.

Address: 0xFC09_C014 (AMPS_SR)

Access: User read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	HP OC	HS OC	SPK 0C	0	0	0	0	0	0	0	AATH DET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-12. Amplifiers Status Register (AMPS_SR)

Table 27-13. AMPS_SR Description

Fields	Description
15–11	Reserved, must be cleared.
10 HPOC	Headphone over-current. Indicates if the headphone amp is flowing excessive current. 0 Headphone Amp is operating normally 1 Excessive current is flowing in headphone amp. Hp_OS_En should be asserted low to turn off the headphone amp output stage. After 25us < Tdelay < 50us Hp_OS_En should be asserted high to turn it back on.

Table 27-13. AMPS_SR Description

Fields	Description
9 HSOC	Handset over-current. Indicates if the handset amp is flowing excessive current. 0 Handset amp is operating normally 1 Excessive current is flowing in handset amp. Hdst_OS_En should be asserted low to turn off the handset amp's output stage. After 25us < Tdelay < 50us Hdst_OS_En should be asserted high to turn it back on.
8 SPKOC	Speaker over-current. Indicates if the speaker amp is flowing excessive current. 0 Speaker amp is operating normally 1 Excessive current is flowing in speaker amp. Spkr_OS_En should be asserted low to turn off the speaker amp's output stage and after 25us < Tdelay < 50us Spkr_OS_En should be asserted high to turn it back on.
7–1	Reserved, must be cleared.
0 AATHDET	Audio amp thermal detect. Indicates if thermal shutdown should be enforced. 0 Temperature is below thermal shutdown limit 1 Temperature is above thermal shutdown limit. Driver amplifiers should be powered down.

27.4 Functional Description

27.4.1 Voice Codec Interrupts

The voice codec has three interrupt vectors.

- Transmit/receive voice codec interrupt
- Receive overrun error interrupt
- Transmit underrun error interrupt
- General voice codec interrupt (logical OR of the above three)

27.4.1.1 Voice Codec Interrupt (VCI)

The voice codec has a single interrupt for input and output paths. It indicates that

- CODEC_TX is empty and a new sample can be output to the DAC.
- CODEC_RX contains a new input sample from the ADC.

The VCI interrupt is enabled by setting CODEC_CR[VCIE]. CODEC_SR[VCRF, VCTE] are set simultaneously with the activation of the VCI interrupt. The interrupt request is cleared when the CODEC_RX register is read and CODEC_TX register is written (in any sequence).

27.4.1.2 Voice Codec Receive Overrun Error Interrupt (VROE)

The interrupt is activated when a new sample is received from the codec section while the previous received sample in CODEC_RX has not been read yet. In this case, the previous received sample is overwritten in CODEC_RX. The interrupt is enabled when CODEC_CR[VCIXE] is set. CODEC_SR[VROE] is set simultaneously with the activation of the VROE interrupt. This interrupt is cleared by reading CODEC_SR with VROE set followed by reading CODEC_RX or by setting CODEC_CR[VCRST]. Clearing CODEC_CR[VCEN] does not affect VROE.

NOTE

The overrun interrupt is only an exception indicator. When there is an overrun situation do not depend on the data in CODEC_RX to be correct.

27.4.1.3 Voice Codec Transmit Underrun Error Interrupt (VTUE)

The interrupt is set when a sample has to be transmitted to the codec section while tCODEC_TX has not yet been written. In this case, the previous received sample is retransmitted to the DAC. The interrupt is enabled when CODEC_CR[VCOXE] is set. VTUE is cleared by reading the CODEC_SR with VTUE set followed by writing CODEC_TX or by setting CODEC_CR[VCRST]. Clearing CODEC_CR[VCEN] does not affect VTUE. CODEC_SR[VTUE] is set simultaneously with the activation of the VTUE interrupt.

NOTE

The underrun interrupt is only an exception indicator. When there is an underrun situation do not count on the data in CODEC_TX.

27.4.1.4 General Voice Codec Interrupt

This device also contains a logical OR of the other three interrupt sources. To determine which of the four situations (transmit empty, receive full, overrun, or underrun) caused this interrupt, read CODEC_SR.

27.4.2 Audio Amplifiers Operation

The audio amplifier analog module consists of three driver amplifiers, a microphone amplifier, and a common mode reference amplifier. As detailed in Section 27.1, “Introduction,” the driver amplifiers provide the power gain to the output signal of DAC to drive loads up to 4-Ω speaker, 8-Ω handset, and 16-Ω headphone amplifiers. The microphone amplifier boosts the weak signal coming from the mic to optimize the ADC performance. The common mode reference amplifier provides an alternate common mode voltage (other than the CM voltage used by the codec) to the driver amplifiers for better performance.

27.4.2.1 Speaker Amplifier

The speaker amplifier is a differential-input/differential-output VGA. It takes its input from the digital to audio converter (DAC) and drives the speaker. It can drive loads down to 4 Ω with a maximum power of 500mW. Table 27-14 specifies the different gain settings of speaker amplifier. The gain control is non-linear and tailored for how people perceive an increase in volume. There is also a mute control with an attenuation of >80dB. All the driver amplifiers have the same volume control.

Table 27-14. Gain Settings for Speaker and Handset Amplifiers

Gain Control Bits			Gain	
Dr_vol <2>	Dr_vol <1>	Dr_vol <0>	V/V	dB
0	0	0	1	0
0	0	1	0.005	–46

Table 27-14. Gain Settings for Speaker and Handset Amplifiers (continued)

Gain Control Bits			Gain	
Dr_vol <2>	Dr_vol <1>	Dr_vol <0>	V/V	dB
0	1	0	0.083	–21.6
0	1	1	0.5	–6
1	0	0	1	0
1	0	1	1.58	4
1	1	0	2	6
1	1	1	2	6

The load of the speaker amplifier can vary depending on your design and which speakers are used. The speaker amplifier can handle up to 300-pF load capacitor for unsustained oscillations at the output. The amplifiers are designed to suppress click/pop when they are enabled (powered up) or disabled (powered down). However, no click/pop suppression occurs during supply ramp-up.

The speaker amplifier is a class-AB output stage that consumes approximately 825 uA of quiescent current and can deliver maximum currents of 500 mA when needed. Due to such high currents flowing in the circuit, electro-migration (EM) and thermal heating of the die are a major concern. Also during a short-circuit condition, the output currents can go to a few amperes if not checked and might lead to thermal runaway.

To protect the amplifier from shorts, an over-current detection and protection circuitry is incorporated in the amplifier. This circuit detects when the current goes above a level that cannot occur during normal operation. The current is clipped to a maximum-allowed level and the amplifier is turned off within 1 μs.

There is a lot of heat generated on-chip due to the power dissipated in the amplifier when it is delivering high power to the load. To keep the temperatures from rising to alarming levels, there is a thermal protection circuit embedded in the processor. The circuit shuts the amplifiers down in case the temperature rises above specified levels as detailed in Section 27.3.11, “Amplifiers Status Register (AMPS_SR).” The amplifiers are turned back on when the temperature falls below a certain level.

Due to the high currents, another issue that can stop the process is the parasitic resistance seen in the output and supply paths. There can be substantial ground bounce and supply droop, especially at high levels. The amplifier supplies are thus kept different from the rest of the codec and the output transistors are placed as close to the pads as possible. Due to the supply droop and GND bounce, the headroom is a major concern at high swings and thus, the common mode voltage should be at mid-supply for optimal performance. However, the codec uses a slightly lower common-mode voltage. There is an option provided to use a slightly higher value of common-mode voltage if required.

27.4.2.2 Handset Amplifier

Handset amplifier is essentially same as the speaker amplifier. The only difference is that the maximum load supported is 8 Ω. The gain settings are the same as speaker amplifier and controlled by the same gain register. Therefore, if you switch from handset to speaker or vice-versa, the volume remains the same. The thermal limit is set to the same as the speaker amplifier. However, with a reduced load, it is less likely to be a problem for the handset amplifier. The over-current limit is set to half of that of speaker amplifier.

27.4.2.3 Headphone Amplifier

The headphone amplifier is a differential-input, single-ended-output VGA. It takes its input from the digital-to-audio converter (DAC) and drives a mono headphone. It can drive loads down to 16 Ω with a maximum power of 31.25 mW. Connect one end of the load to the amplifier and the other end to ground.

The DC operating point of the amplifier output sits close to mid-supply to support the maximum possible dynamic range. This leads to a DC voltage drop across the load that can amount to approximately 80 mA of DC current through the load. Thus, the amplifier's output is AC-coupled to the load to stop the DC current. A 100 µF capacitor is suggested as a decoupling cap to get approximately 100 Hz of high-pass corner frequency.

Unlike the speaker and handset amplifiers, the headphone amplifier can see a wide range of loads that can range from a 16-Ω resister with few picofarad capacitor to ground to a couple nanofarad capacitor to ground with >10 kΩ resistive load. To guarantee stability of the amplifier for this wide range of loads, a dummy load of approximately 50 Ω is always connected at the output of the amplifier to GND. Again, to stop DC current flow through this resistor, a DC blocking cap is required. For the dummy output, a 100 nF decoupling capacitor is suggested, as it works well for stability and also provides high-series impedance at signal frequencies.

The mute control of headphone amplifier is the same as the speaker and handset amplifiers. Headphone amplifier also supports over-current protection and click and pop-suppression. The gain settings for the headphone amplifier are slightly different from those of the speaker amplifier because the swing in the headphone amp is limited due to its single-ended output. The gain settings for the headphone amplifier are shown in Table 27-15.

Table 27-15. Gain Settings for Headphone Amplifier

Gain Control Bits			Gain	
Dr_vol <2>	Dr_vol <1>	Dr_vol <0>	V/V	dB
0	0	0	0.5	–6
0	0	1	0.005	–46
0	1	0	0.083	–21.6
0	1	1	0.25	–12
1	0	0	0.5	–6
1	0	1	0.79	–2
1	1	0	1	0
1	1	1	1	0

27.4.2.4 Microphone Amplifier

The microphone amplifier is a differential-input/differential-output variable gain amp (VGA). It takes single-ended or differential input from a microphone and gives differential output to the analog-to-digital converter (ADC).

For a single-ended input, the negative input should be connected to ground (GND).If you want to use an external amplifier, the internal microphone amplifier can be bypassed. Also, the inputs are AC-coupled to the microphone. The common mode DC levels are derived from the common mode voltage generated in the reference block (Vag). This configuration ensures that the DC-level at the amplifier inputs is governed by the reference while its AC-input comes from the microphone.

The gain setting of microphone amplifier is specified in Table 27-16. It also has a mute control with attenuation >80 dB. As seen in Table 27-16, the input impedance of the amplifier changes with the gain setting with a minimum of 1.82 kΩ at gain = 99V/V.

The DC coupling caps (C_{DC}) and the input resistor (R_{IN}) form a high-pass filter with corner frequency of f_{HP} equaling $1/(2\times R_{IN}\times C_{DC})$. When R_{IN} equals 1.82 kΩ for a corner frequency of 125 Hz and taking into account the resistor variation across corners a C_{DC} of 1 µF is suggested.

If you bypass the microphone amplifier and an external microphone amplifier is used, a C_{DC} of 100 nF is suggested.

Table 27-16. Gain settings for Microphone Amplifiers

Gain Control Bits			Gain		R_i (Ω)	R_f (Ω)
g <2>	g <1>	g <0>	V/V	dB		
0	0	0	1	0	20	20
0	0	1	2	6	10	20
0	1	0	3	9.56	20	60
0	1	1	6	15.56	10	60
1	0	0	10	20	6	60
1	0	1	16	24	3.75	60
1	1	0	30	29.56	2	60
1	1	1	99	39.92	1.82	180

Chapter 28
Fast Ethernet Controllers (FEC0 and FEC1)

28.1 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 low pin-count 10/100 Mbps reduced MII and 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

28.1.1 Overview

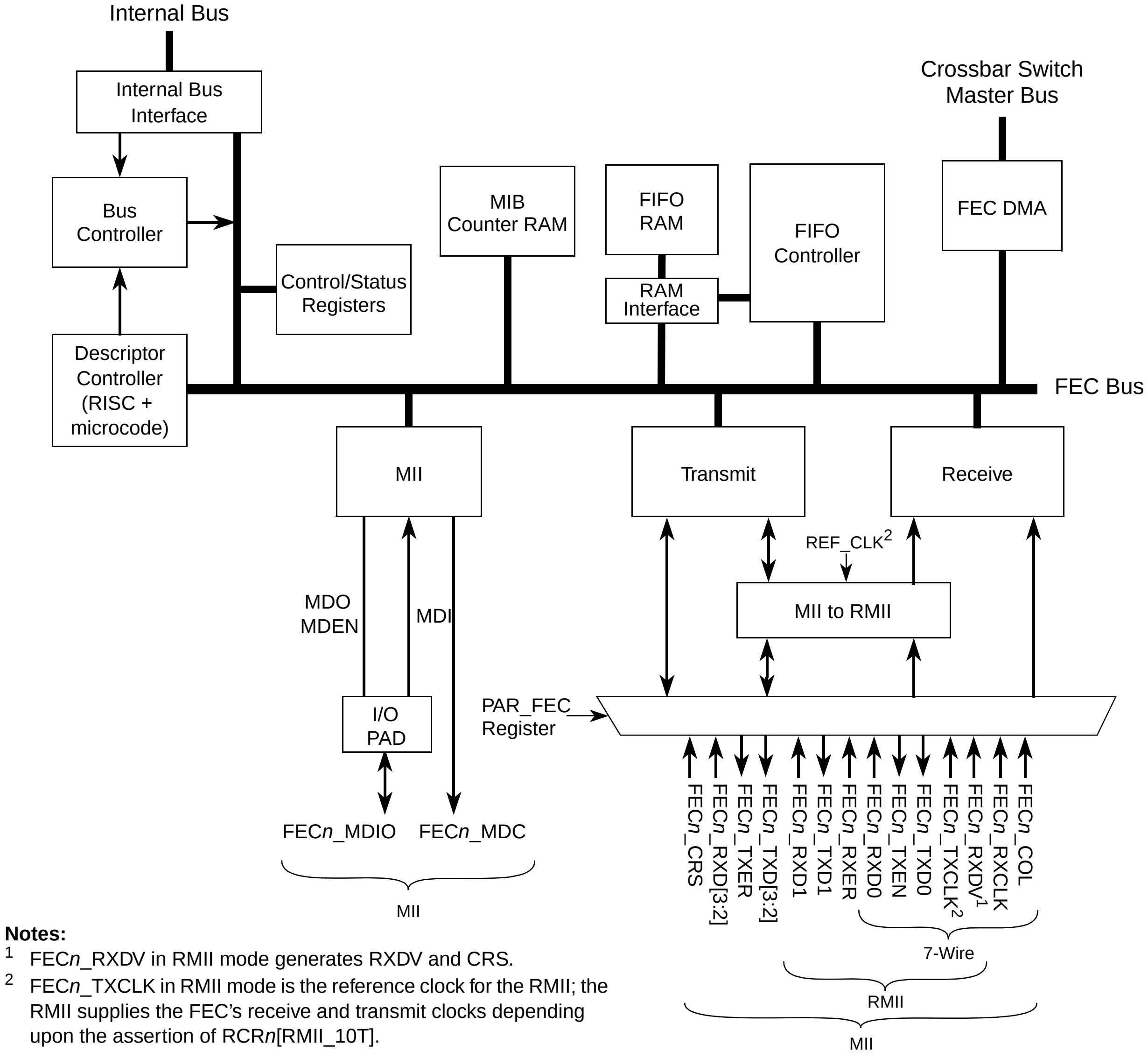
The Ethernet media access controller (MAC) supports 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FECs support five different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FECs support the 10/100 Mbps MII, 10/100 Mbps reduced MII, and the 10 Mbps-only 7-wire interface.

NOTE

The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to ”) prior to configuring the FECs.

28.1.2 Block Diagram

Figure 28-1 shows the block diagram of a single FEC. The FECs are implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.



Notes:
[1] FEC*n*_RXDV in RMII mode generates RXDV and CRS.
[2] FEC*n*_TXCLK in RMII mode is the reference clock for the RMII; the RMII supplies the FEC's receive and transmit clocks depending upon the assertion of RCR*n*[RMII_10T].

Figure 28-1. FEC*n* Block Diagram

The descriptor controller is a RISC-based controller providing these functions in the FECs:

- Initialization (those internal registers not initialized by you or hardware)
- High level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

NOTE

DMA references in this section refer to the FEC's DMA engine. This DMA engine transfers FEC data only and is not related to the eDMA controller described in Chapter 17, "Enhanced Direct Memory Access (eDMA)," nor to the DMA timers described in Chapter 32, "DMA Timers (DTIM0–DTIM3)."

The RAM is the focal point of all data flow in the Fast Ethernet controller and divides into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR*n* register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block, and receive data flows from the receive block into the receive FIFO.

You control the FECs by writing into control registers located in each block. The CSR (control and status registers) block provides global control (Ethernet reset and enable) and interrupt managing registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the FEC*n*_MDC (management data clock) and FEC*n*_MDIO (management data input/output) lines of the MII interface.

The FEC DMA block (not to be confused with the device's eDMA controller) provides multiple channels allowing transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

The transmit and receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The message information block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC, but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters. See Section 28.4.1, "MIB Block Counters Memory Map," for more information.

28.1.3 Features

The FECs incorporate the following features:

- Support for five different Ethernet physical interfaces:
  - 100-Mbps IEEE 802.3 MII
  - 10-Mbps IEEE 802.3 MII
  - 100-Mbps reduced media independent interface (RMII)
  - 10-Mbps reduced media independent interface (RMII)
  - 10-Mbps 7-wire interface (industry standard)

NOTE

This device's FEC1 supports only RMII mode. Also, the signals used in MII and 7-wire modes on FEC0 are multiplexed with FEC1's RMII signals. Therefore, both FECs may be used in RMII modes simultaneously, or FEC0 can be used in MII or 7-wire mode while FEC1 is disabled.

- 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

28.2 Modes of Operation

The primary operational modes are described in this section.

28.2.1 Full and Half Duplex Operation

Full duplex mode is for use on point-to-point links between switches or end node to switch. Half duplex mode works in connections between an end node and a repeater or between repeaters. TCR*n*[FDEN] controls duplex mode selection.

When configured for full duplex mode, flow control may be enabled. Refer to the TCR*n*[RFC_PAUSE,TFC_PAUSE] bits, the RCR*n*[FCE] bit, and Section 28.5.11, “Full Duplex Flow Control,” for more details.

28.2.2 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in Section 28.5.6, “Network Interface Options.”

28.2.2.1 10 Mbps and 100 Mbps MII Interface

The IEEE 802.3 standard defines the media independent interface (MII) for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by setting RCR*n*[MII_MODE].

FEC*n*_TXCLK and FEC*n*_RXCLK pins driven by the external transceiver determine the operation speed. The transceiver auto-negotiates the speed or software controls it via the serial management interface (FEC*n*_MDC/FEC*n*_MDIO pins) to the transceiver. Refer to the MMFR*n* and MSCR*n* register descriptions, as well as the section on the MII, for a description of how to read and write registers in the transceiver via this interface.

28.2.2.2 10 Mbps and 100 Mbps RMII Interface

The reduced media independent interface (RMII) is a low cost alternative to the IEEE 802.3 MII standard. This interface provides the functionality of the MII interface on a total of 8 pins instead of 18. The RMII interface for 10/100 Ethernet MAC-PHY interface was defined by an industry consortium and is not currently included in the IEEE 802.3 standard. The PAR_FEC register in the pin multiplexing and control module controls this functionality which is reflected in the read-only RCR*n*[RMII_MODE] bit. The RCR*n*[RMII_10T] bit determines the speed of operation. The reference clock for RMII is always 50 MHz, but this clock can be divided by 10 within the RCR register to support 10 Mbps operation. The PHY must be configured accordingly. See ,” for more details on the PAR_FEC register.

28.2.2.3 10 Mpbs 7-Wire Interface Operation

The FECs support 7-wire interface used by many 10 Mbps Ethernet transceivers. The RCR[MII_MODE] bit controls this functionality. If this bit is cleared, MII mode is disabled and the 10 Mbps 7-wire mode is enabled.

28.2.3 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 28.5.9, “Ethernet Address Recognition.”

28.2.4 Internal Loopback

Internal loopback mode is selected via RCR*n*[LOOP]. Loopback mode is discussed in detail in Section 28.5.14, “MII Internal and External Loopback.”

28.3 External Signal Description

Table 28-1 describes the various FEC signals, as well as indicating which signals work in available modes.

Table 28-1. FEC Signal Descriptions

Signal Name	MII	7-wire	RMII	Description
FEC_COL	X	X	—	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.
FEC_CRS	X	—	—	Carrier sense. When asserted, indicates transmit or receive medium is not idle. In RMII mode, this signal is present on the FEC_RXDV pin.
FEC_MDC	X	—	—	Output clock provides a timing reference to the PHY for data transfers on the FEC_MDIO signal.
FEC_MDIO	X	—	—	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC_MDC. This signal is an input after reset. When the FEC operates in 10Mbps 7-wire interface mode, this signal should be connected to VSS.

Table 28-1. FEC Signal Descriptions (continued)

Signal Name	MII	7-wire	RMII	Description
FEC_RXCLK	X	X	—	Provides a timing reference for FEC_RXDV, FEC_RXD[3:0], and FEC_RXER.
FEC_RXDV	X	X	X	Asserting the FEC_RXDV input indicates PHY has valid nibbles present on the MII. FEC_RXDV must remain asserted from the first recovered nibble of the frame through to the last nibble. Assertion of FEC_RXDV must start no later than the SFD and exclude any EOF. In RMII mode, this pin also generates the CRS signal.
FEC_RXD0	X	X	X	This pin contains the Ethernet input data transferred from PHY to the media-access controller when FEC_RXDV is asserted.
FEC_RXD1	X	—	X	This pin contains the Ethernet input data transferred from PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXD[3:2]	X	—	—	These pins contain the Ethernet input data transferred from PHY to the media access controller when FEC_RXDV is asserted.
FEC_RXER	X	—	X	When asserted with FEC_RXDV, indicates PHY detects an error in the current frame. When FEC_RXDV is not asserted, FEC_RXER has no effect.
FEC_TXCLK	X	X	X	Input clock which provides a timing reference for FEC_TXEN, FEC_TXD[3:0] and FEC_TXER. In RMII mode, this signal is the reference clock for receive, transmit, and the control interface.
FEC_TXD0	X	X	X	The serial output Ethernet data and only valid during the assertion of FEC_TXEN.
FEC_TXD1	X	—	X	This pin contains the serial output Ethernet data and valid only during assertion of FEC_TXEN.
FEC_TXD[3:2]	X	—	—	These pins contain the serial output Ethernet data and valid only during assertion of FEC_TXEN.
FEC_TXEN	X	X	X	Indicates when valid nibbles are present on the MII. This signal is asserted with the first nibble of a preamble and is negated before the first FEC_TXCLK following the final nibble of the frame.
FEC_TXER	X	—	—	When asserted for one or more clock cycles while FEC_TXEN is also asserted, PHY sends one or more illegal symbols. FEC_TXER has no effect at 10 Mbps or when FEC_TXEN is negated.

28.4 Memory Map/Register Definition

The FECs are programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs control operation modes and extract global status information. The descriptors pass data buffers and related buffer information between the hardware and software.

Each FEC implementation requires a 1-Kbyte memory map space, which is divided into two sections of 512 bytes each for:

- Control/status registers
- Event/statistic counters held in the MIB block

Table 28-2 defines the top level memory map.

Table 28-2. Module Memory Map

Address	Function
0xFC03_0000 – FC03_01FF	FEC0 Control/Status Registers
0xFC03_0200 – FC03_02FF	FEC0 MIB Block Counters

Table 28-2. Module Memory Map (continued)

Address	Function
0xFC03_4000 – FC03_41FF	FEC1 Control/Status Registers
0xFC03_4200 – FC03_42FF	FEC1 MIB Block Counters

Table 28-3 shows the FEC register memory map.

Table 28-3. FEC Register Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
FEC0 FEC1					
0xFC03_0004 0xFC03_4004	Interrupt Event Register (EIR*n*)	32	R/W	0x0000_0000	28.4.2/28-12
0xFC03_0008 0xFC03_4008	Interrupt Mask Register (EIMR*n*)	32	R/W	0x0000_0000	28.4.3/28-13
0xFC03_0010 0xFC03_4010	Receive Descriptor Active Register (RDAR*n*)	32	R/W	0x0000_0000	28.4.4/28-14
0xFC03_0014 0xFC03_4014	Transmit Descriptor Active Register (TDAR*n*)	32	R/W	0x0000_0000	28.4.5/28-15
0xFC03_0024 0xFC03_4024	Ethernet Control Register (ECR*n*)	32	R/W	0xF000_0000	28.4.6/28-15
0xFC03_0040 0xFC03_4040	MII Management Frame Register (MMFR*n*)	32	R/W	Undefined	28.4.7/28-16
0xFC03_0044 0xFC03_4044	MII Speed Control Register (MSCR*n*)	32	R/W	0x0000_0000	28.4.8/28-18
0xFC03_0064 0xFC03_4064	MIB Control/Status Register (MIBC*n*)	32	R/W	0x0000_0000	28.4.9/28-19
0xFC03_0084 0xFC03_4084	Receive Control Register (RCR*n*)	32	R/W	0x05EE_0001	28.4.10/28-19
0xFC03_00C4 0xFC03_40C4	Transmit Control Register (TCR*n*)	32	R/W	0x0000_0000	28.4.11/28-21
0xFC03_00E4 0xFC03_40E4	Physical Address Low Register (PALR*n*)	32	R/W	Undefined	28.4.12/28-22
0xFC03_00E8 0xFC03_40E8	Physical Address High Register (PAUR*n*)	32	R/W	See Section	28.4.13/28-23
0xFC03_00EC 0xFC03_40EC	Opcode/Pause Duration (OPD*n*)	32	R/W	See Section	28.4.14/28-23
0xFC03_0118 0xFC03_4118	Descriptor Individual Upper Address Register (IAUR*n*)	32	R/W	Undefined	28.4.15/28-24
0xFC03_011C 0xFC03_411C	Descriptor Individual Lower Address Register (IALR*n*)	32	R/W	Undefined	28.4.16/28-24

Table 28-3. FEC Register Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
FEC0 FEC1					
0xFC03_0120 0xFC03_4120	Descriptor Group Upper Address Register (GAUR*n*)	32	R/W	Undefined	28.4.17/28-25
0xFC03_0124 0xFC03_4124	Descriptor Group Lower Address Register (GALR*n*)	32	R/W	Undefined	28.4.18/28-25
0xFC03_0144 0xFC03_4144	Transmit FIFO Watermark (TFWR*n*)	32	R/W	0x0000_0000	28.4.19/28-26
0xFC03_014C 0xFC03_414C	FIFO Receive Bound Register (FRBR*n*)	32	R	0x0000_0600	28.4.20/28-26
0xFC03_0150 0xFC03_4150	FIFO Receive FIFO Start Register (FRSR*n*)	32	R	0x0000_0500	28.4.21/28-27
0xFC03_0180 0xFC03_4180	Pointer to Receive Descriptor Ring (ERDSR*n*)	32	R/W	Undefined	28.4.22/28-27
0xFC03_0184 0xFC03_4184	Pointer to Transmit Descriptor Ring (ETDSR*n*)	32	R/W	Undefined	28.4.23/28-28
0xFC03_0188 0xFC03_4188	Maximum Receive Buffer Size (EMRBR*n*)	32	R/W	Undefined	28.4.24/28-28

28.4.1 MIB Block Counters Memory Map

The MIB counters memory map (Table 28-4) 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 28-4. MIB Counters Memory Map

Address	Register
FEC0 FEC1	
0xFC03_0200 0xFC03_4200	Count of frames not counted correctly (RMON_T_DROP*n*)
0xFC03_0204 0xFC03_4204	RMON Tx packet count (RMON_T_PACKETS*n*)
0xFC03_0208 0xFC03_4208	RMON Tx broadcast packets (RMON_T_BC_PKT*n*)
0xFC03_020C 0xFC03_420C	RMON Tx multicast packets (RMON_T_MC_PKT*n*)
0xFC03_0210 0xFC03_4210	RMON Tx packets with CRC/align error (RMON_T_CRC_ALIGN*n*)
0xFC03_0214 0xFC03_4214	RMON Tx packets < 64 bytes, good CRC (RMON_T_UNDERSIZE*n*)
0xFC03_0218 0xFC03_4218	RMON Tx packets > MAX_FL bytes, good CRC (RMON_T_OVERSIZE*n*)
0xFC03_021C 0xFC03_421C	RMON Tx packets < 64 bytes, bad CRC (RMON_T_FRAG*n*)
0xFC03_0220 0xFC03_4220	RMON Tx packets > MAX_FL bytes, bad CRC (RMON_T_JAB*n*)
0xFC03_0224 0xFC03_4224	RMON Tx collision count (RMON_T_COL*n*)
0xFC03_0228 0xFC03_4228	RMON Tx 64 byte packets (RMON_T_P64*n*)
0xFC03_022C 0xFC03_422C	RMON Tx 65 to 127 byte packets (RMON_T_P65TO127*n*)
0xFC03_0230 0xFC03_4230	RMON Tx 128 to 255 byte packets (RMON_T_P128TO255*n*)
0xFC03_0234 0xFC03_4234	RMON Tx 256 to 511 byte packets (RMON_T_P256TO511*n*)
0xFC03_0238 0xFC03_4238	RMON Tx 512 to 1023 byte packets (RMON_T_P512TO1023*n*)
0xFC03_023C 0xFC03_423C	RMON Tx 1024 to 2047 byte packets (RMON_T_P1024TO2047*n*)
0xFC03_0240 0xFC03_4240	RMON Tx packets with > 2048 bytes (RMON_T_P_GTE2048*n*)
0xFC03_0244 0xFC03_4244	RMON Tx Octets (RMON_T_OCTETS*n*)
0xFC03_0248 0xFC03_4248	Count of transmitted frames not counted correctly (IEEE_T_DROP*n*)

Table 28-4. MIB Counters Memory Map (continued)

Address	Register
FEC0 FEC1	
0xFC03_024C 0xFC03_424C	Frames transmitted OK (IEEE_T_FRAME_OK*n*)
0xFC03_0250 0xFC03_4250	Frames transmitted with single collision (IEEE_T_1COL*n*)
0xFC03_0254 0xFC03_4254	Frames transmitted with multiple collisions (IEEE_T_MCOL*n*)
0xFC03_0258 0xFC03_4258	Frames transmitted after deferral delay (IEEE_T_DEF*n*)
0xFC03_025C 0xFC03_425C	Frames transmitted with late collision (IEEE_T_LCOL*n*)
0xFC03_0260 0xFC03_4260	Frames transmitted with excessive collisions (IEEE_T_EXCOL*n*)
0xFC03_0264 0xFC03_4264	Frames transmitted with Tx FIFO underrun (IEEE_T_MACERR*n*)
0xFC03_0268 0xFC03_4268	Frames transmitted with carrier sense error (IEEE_T_CSERR*n*)
0xFC03_026C 0xFC03_426C	Frames transmitted with SQE error (IEEE_T_SQE*n*)
0xFC03_0270 0xFC03_4270	Flow control pause frames transmitted (IEEE_T_FDXFC*n*)
0xFC03_0274 0xFC03_4274	Octet count for frames transmitted without error (IEEE_T_OCTETS_OK*n*)
0xFC03_0280 0xFC03_4280	Count of received frames not counted correctly (RMON_R_DROP*n*)
0xFC03_0284 0xFC03_4284	RMON Rx packet count (RMON_R_PACKETS*n*)
0xFC03_0288 0xFC03_4288	RMON Rx broadcast packets (RMON_R_BC_PKT*n*)
0xFC03_028C 0xFC03_428C	RMON Rx multicast packets (RMON_R_MC_PKT*n*)
0xFC03_0290 0xFC03_4290	RMON Rx packets with CRC/Align error (RMON_R_CRC_ALIGN*n*)
0xFC03_0294 0xFC03_4294	RMON Rx packets < 64 bytes, good CRC (RMON_R_UNDERSIZE*n*)
0xFC03_0298 0xFC03_4298	RMON Rx packets > MAX_FL bytes, good CRC (RMON_R_OVERSIZE*n*)
0xFC03_029C 0xFC03_429C	RMON Rx packets < 64 bytes, bad CRC (RMON_R_FRAG*n*)

Table 28-4. MIB Counters Memory Map (continued)

Address	Register
FEC0 FEC1	
0xFC03_02A0 0xFC03_42A0	RMON Rx packets > MAX_FL bytes, bad CRC (RMON_R_JAB*n*)
0xFC03_02A4 0xFC03_42A4	Reserved (RMON_R_RESVD_0*n*)
0xFC03_02A8 0xFC03_42A8	RMON Rx 64 byte packets (RMON_R_P64*n*)
0xFC03_02AC 0xFC03_42AC	RMON Rx 65 to 127 byte packets (RMON_R_P65TO127*n*)
0xFC03_02B0 0xFC03_42B0	RMON Rx 128 to 255 byte packets (RMON_R_P128TO255*n*)
0xFC03_02B4 0xFC03_42B4	RMON Rx 256 to 511 byte packets (RMON_R_P256TO511*n*)
0xFC03_02B8 0xFC03_42B8	RMON Rx 512 to 1023 byte packets (RMON_R_P512TO1023*n*)
0xFC03_02BC 0xFC03_42BC	RMON Rx 1024 to 2047 byte packets (RMON_R_P1024TO2047*n*)
0xFC03_02C0 0xFC03_42C0	RMON Rx packets with > 2048 bytes (RMON_R_P_GTE2048*n*)
0xFC03_02C4 0xFC03_42C4	RMON Rx octets (RMON_R_OCTETS*n*)
0xFC03_02C8 0xFC03_42C8	Count of received frames not counted correctly (IEEE_R_DROP*n*)
0xFC03_02CC 0xFC03_42CC	Frames received OK (IEEE_R_FRAME_OK*n*)
0xFC03_02D0 0xFC03_42D0	Frames received with CRC error (IEEE_R_CRC*n*)
0xFC03_02D4 0xFC03_42D4	Frames received with alignment error (IEEE_R_ALIGN*n*)
0xFC03_02D8 0xFC03_42D8	Receive FIFO overflow count (IEEE_R_MACERR*n*)
0xFC03_02DC 0xFC03_42DC	Flow control pause frames received (IEEE_R_FDXFC*n*)
0xFC03_02E0 0xFC03_42E0	Octet count for frames received without error (IEEE_R_OCTETS_OK*n*)

28.4.2 Ethernet Interrupt Event Registers (EIR0 & EIR1)

When an event occurs that sets a bit in EIR*n*, an interrupt occurs if the corresponding bit in the interrupt mask register (EIMR) is also set. Writing a 1 to an EIR bit clears it; writing 0 has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts which may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC, and RL. Interrupts resulting from internal errors are HBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters:

- HBERR - IEEE_T_SQE
- BABR - RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT - RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)
- LATE_COL - IEEE_T_LCOL
- COL_RETRY_LIM - IEEE_T_EXCOL
- XFIFO_UN - IEEE_T_MACERR

Software may choose to mask off these interrupts because these errors are visible to network management via the MIB counters.

Address: 0xFC03_0004 (EIR0)
0xFC03_4004 (EIR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-2. Ethernet Interrupt Event Register (EIR*n*)

Table 28-5. EIR*n* Field Descriptions

Field	Description
31 HBERR	Heartbeat error. Indicates TCR*n*[HBC] is set and that the COL input was not asserted within the heartbeat window following a transmission.
30 BABR	Babbling receive error. Indicates a frame was received with length in excess of RCR*n*[MAX_FL] bytes.
29 BABT	Babbling transmit error. Indicates the transmitted frame length exceeds RCR*n*[MAX_FL] bytes. Usually this condition is caused by a frame that is too long is placed into the transmit data buffer(s). Truncation does not occur.

Table 28-5. EIR*n* Field Descriptions (continued)

Field	Description
28 GRA	Graceful stop complete. Indicates the graceful stop is complete. During graceful stop the transmitter is placed into a pause state after completion of the frame currently being transmitted. This bit is set by one of three conditions: 1) A graceful stop initiated by the setting of the TCR*n*[GTS] bit is now complete. 2) A graceful stop initiated by the setting of the TCR*n*[TFC_PAUSE] bit is now complete. 3) A graceful stop initiated by the reception of a valid full duplex flow control pause frame is now complete. Refer to Section 28.5.11, "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*n*[ETHER_EN] is cleared, halting frame processing by the FEC. When this occurs, software needs to ensure that the FIFO controller and DMA also soft reset.
21 LC	Late collision. Indicates a collision occurred beyond the collision window (slot time) in half duplex mode. The frame truncates with a bad CRC and the remainder of the frame is discarded.
20 RL	Collision retry limit. Indicates a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted and transmission of the next frame commences. This error can only occur in half duplex mode.
19 UN	Transmit FIFO underrun. Indicates the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.
18–0	Reserved, must be cleared.

28.4.3 Interrupt Mask Registers (EIMR0 & EIMR1)

The EIMR*n* registers control which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. A hardware reset clears this register. If the corresponding bits in the EIR*n* and EIMR*n* registers are set, an interrupt is generated. The interrupt signal remains asserted until a 1 is written to the EIR*n* bit (write 1 to clear) or a 0 is written to the EIMR*n* bit.

Address: 0xFC03_0008 (EIMR0)
0xFC03_4008 (EIMR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-3. Ethernet Interrupt Mask Register (EIMR*n*)

Table 28-6. EIMR*n* Field Descriptions

Field	Description
31–19 See Figure 28-3 and Table 28-5	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR*n* register. The corresponding EIMR*n* bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR*n* samples the signal generated by the interrupting source. The corresponding EIR*n* bit reflects the state of the interrupt signal even if the corresponding EIMR*n* bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
18–0	Reserved, must be cleared.

28.4.4 Receive Descriptor Active Registers (RDAR0 & RDAR1)

RDAR*n* is a command register, written by the user, indicating the receive descriptor ring is updated (the driver produced empty receive buffers with the empty bit set).

When the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and processes receive frames (provided ECR*n*[ETHER_EN] is also set). After the FEC polls a receive descriptor whose empty bit is not set, FEC clears the RDAR bit and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors are placed into the receive descriptor ring.

The RDAR registers are cleared at reset and when ECR*n*[ETHER_EN] is cleared.

Address: 0xFC03_0010 (RDAR0)
0xFC03_4010 (RDAR1)

Access: User read/write

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

Figure 28-4. Receive Descriptor Active Register (RDAR*n*)

Table 28-7. RDAR*n* Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 RDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional empty descriptors remain in the receive ring. Also cleared when ECR*n*[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

28.4.5 Transmit Descriptor Active Registers (TDAR0 & TDAR1)

The TDAR*n* are command registers which the user writes to indicate the transmit descriptor ring is updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

When the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the FEC polls the transmit descriptor ring and processes transmit frames (provided ECR*n*[ETHER_EN] is also set). After the FEC polls a transmit descriptor that is a ready bit not set, FEC clears the TDAR bit and ceases transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors are placed into the transmit descriptor ring.

The TDAR*n* register is cleared at reset, when ECR*n*[ETHER_EN] is cleared, or when ECR*n*[RESET] is set.

Address: 0xFC03_0014 (TDAR0)
0xFC03_4014 (TDAR1)

Access: User read/write

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

Figure 28-5. Transmit Descriptor Active Register (TDAR*n*)

Table 28-8. TDAR*n* Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 TDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional ready descriptors remain in the transmit ring. Also cleared when ECR*n*[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

28.4.6 Ethernet Control Registers (ECR0 & ECR1)

ECR*n* is a read/write user register, though hardware may alter fields in this register as well. The ECR*n* enables/disables the FEC.

Address: 0xFC03_0024 (ECR0)
0xFC03_4024 (ECR1)

Access: User read/write

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

Figure 28-6. Ethernet Control Register (ECR*n*)

Table 28-9. ECR*n* 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*n*[RESET] is set by software, in which case ETHER_EN is cleared • An error condition causes the EIR*n*[EBERR] bit to set, in which case ETHER_EN is cleared
0 RESET	When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. ECR*n*[ETHER_EN] is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately eight internal bus clock cycles after this bit is set.

28.4.7 MII Management Frame Registers (MMFR0 & MMFR1)

The MMFR*n* is user-accessible and does not reset to a defined value. The MMFR*n* registers are used to communicate with the attached MII compatible PHY device(s), providing read/write access to their MII registers. Performing a write to the MMFR*n* causes a management frame to be sourced unless the MSCR*n* is programmed to 0. If MSCR*n* is cleared while MMFR*n* is written and then MSCR*n* is written with a non-zero value, an MII frame is generated with the data previously written to the MMFR*n*. This allows MMFR*n* and MSCR*n* to be programmed in either order if MSCR*n* is currently zero.

Address: 0xFC03_0040 (MMFR0)
0xFC03_4040 (MMFR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ST		OP		PA					RA					TA		DATA															
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 28-7. MII Management Frame Register (MMFR*n*)

Table 28-10. MMFR*n* Field Descriptions

Field	Description
31–30 ST	Start of frame delimiter. These bits must be programmed to 0b01 for a valid MII management frame.
29–28 OP	Operation code. 00 Write frame operation, but not MII compliant. 01 Write frame operation for a valid MII management frame. 10 Read frame operation for a valid MII management frame. 11 Read frame operation, but not MII compliant.
27–23 PA	PHY address. This field specifies one of up to 32 attached PHY devices.
22–18 RA	Register address. This field specifies one of up to 32 registers within the specified PHY device.
17–16 TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
15–0 DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

To perform a read or write operation on the MII Management Interface, write the MMFR*n* register. To generate a valid read or write management frame, ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame is generated, but does not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII Management Interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR*n* register. Writing this pattern causes the control logic to shift out the data in the MMFR*n* register following a preamble generated by the control state machine. During this time, contents of the MMFR*n* register are altered as the contents are serially shifted and are unpredictable if read by the user. After the write management frame operation completes, the MII interrupt is generated. At this time, contents of the MMFR*n* register match the original value written.

To generate an MII management interface read frame (read a PHY register), the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR*n* register (the content of the DATA field is a don’t care). Writing this pattern causes the control logic to shift out the data in the MMFR*n* register following a preamble generated by the control state machine. During this time, contents of the MMFR*n* register are altered as the contents are serially shifted and are unpredictable if read by the user. After the read management frame operation completes, the MII interrupt is generated. At this time, the contents of the MMFR*n* register match the original value written except for the DATA field whose contents are replaced by the value read from the PHY register.

If the MMFR*n* register is written while frame generation is in progress, the frame contents are altered. Software must use the MII interrupt to avoid writing to the MMFR*n* register while frame generation is in progress.

28.4.8 MII Speed Control Registers (MSCR0 & MSCR1)

The MSCR*n* provides control of the MII clock (FEC*n*_MDC pin) frequency and allows a preamble drop on the MII management frame.

Address: 0xFC03_0044 (MSCR0)
0xFC03_4044 (MSCR1)

Access: User read/write

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

Figure 28-8. MII Speed Control Register (MSCRn)

Table 28-11. MSCR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 DIS_PRE	Setting this bit causes the preamble (32 ones) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY device(s) does not require it.
6–1 MII_SPEED	Controls the frequency of the MII management interface clock (FEC*n*_MDC) relative to the internal bus clock. A value of 0 in this field turns off the FEC*n*_MDC and leaves it in low voltage state. Any non-zero value results in the FEC*n*_MDC frequency of 1/(MII_SPEED × 2) of the internal bus frequency.
0	Reserved, must be cleared.

The MII_SPEED field must be programmed with a value to provide an FEC*n*_MDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII_SPEED must be set to a non-zero value to source a read or write management frame. After the management frame is complete, the MSCR*n* register may optionally be set to 0 to turn off the FEC*n*_MDC. The FEC*n*_MDC generated has a 50% duty cycle except when MII_SPEED changes during operation (change takes effect following a rising or falling edge of FEC*n*_MDC).

If the internal bus clock is 25 MHz, programming this register to 0x0000_0005 results in an FEC*n*_MDC as stated the equation below.

$$25\text{ MHz} \times \frac{1}{5 \times 2} = 2.5\text{ MHz} \qquad \textit{Eqn. 28-1}$$

A table showing optimum values for MII_SPEED as a function of internal bus clock frequency is provided below.

Table 28-12. Programming Examples for MSCR*n*

Internal FEC Clock Frequency	MSCR[MII_SPEED]	FEC*n*_MDC frequency
25 MHz	0x5	2.50 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.50 MHz

Table 28-12. Programming Examples for MSCR*n* (continued)

Internal FEC Clock Frequency	MSCR[MII_SPEED]	FEC*n*_MDC frequency
50 MHz	0xA	2.50 MHz
66 MHz	0xE	2.36 MHz

28.4.9 MIB Control Registers (MIBC0 & MIBC1)

The MIBC is a read/write register controlling and observing the state of the MIB block. User software accesses this register if there is a need to disable the MIB block operation. For example, to clear all MIB counters in RAM:

1. Disable the MIB block
2. Clear all the MIB RAM locations
3. Enable the MIB block

The MIB_DIS bit is reset to 1. See Table 28-4 for the locations of the MIB counters.

Address: 0xFC03_0064 (MIBC0)
0xFC03_4064 (MIBC1)

Access: User read/write

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

Figure 28-9. MIB Control Register (MIBC*n*)

Table 28-13. MIBC*n* Field Descriptions

Field	Description
31 MIB_DIS	A read/write control bit. If set, the MIB logic halts and not update any MIB counters.
30 MIB_IDLE	A read-only status bit. If set the MIB block is not currently updating any MIB counters.
29–0	Reserved.

28.4.10 Receive Control Registers (RCR0 & RCR1)

RCR*n* controls the operational mode of the receive block and must be written only when ECR*n*[ETHER_EN] is cleared (initialization time).

Address: 0xFC03_0084 (RCR0)
0xFC03_4084 (RCR1)

Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	RMII_ ECHO	RMII_ LOOP	RMII_ 10T	RMII_ MODE	0	0	FCE	BC_ REJ	PROM	MII_ MODE	DRT	LOOP
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 28-10. Receive Control Register (RCR*n*)

Table 28-14. RCR*n* Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26–16 MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL causes the BABT interrupt to occur. Receive frames longer than MAX_FL causes the BABR interrupt to occur and sets the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed is 1518 or 1522 if VLAN tags are supported.
15–12	Reserved, must be cleared.
11 RMII_ECHO	RMII Echo. Enables RMII echo mode. RMII 2-bit receive data is processed through the RMII receive logic to the FEC and also routes through the RMII's transmit logic to become RMII 2-bit transmit data out. 0 Normal operation. 1 RMII echo mode enabled. **Note:** When RMII_ECHO is set, proper operation is guaranteed only when RMII_MODE = 1, RMII_LOOP = 0 and LOOP = 0.
10 RMII_LOOP	RMII loopback. Enables RMII loopback mode. Causes the MII transmit outputs from the Ethernet controller to loop back to the Ethernet controllers's MII receive inputs through the RMII transmit/receive logic. 0 Normal operation. 1 RMII loopback mode enabled. **Note:** When RMII_LOOP is set, proper operation is guaranteed only when RMII_MODE = 1, RMII_ECHO = 0, LOOP = 0, and TCR*n*[FDEN] = 1.
9 RMII_10T	RMII 10-Base T. Enables 10Mbps mode of the RMII. Determines the clock frequency of the clock source to the FEC logic to support 10/100Mbps operations. 0 100 Mbps operation. The 50 MHz RMII reference clock on FEC_TXCLK is sent to the RMII, while a divided-by-2 version (25 MHz) is sent to the FEC. 1 10 Mbps operation. The 50 MHz RMII reference clock on FEC_TXCLK is divided by 10 (5 MHz) and sent to the RMII, while a divided-by-20 version (2.5 MHz) is sent to the FEC.

Table 28-14. RCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8
RMII_MODE</td><td>RMII Mode. Indicates if the FEC is in RMII or MII/7-wire mode. This is a read-only bit that reflects the status of the MII_MODE bit AND’d with MISCCR[FECM] in the CCM. See ,” for more details.

For FEC0: RMII_MODE = RCR0[MII_MODE] && MISCCR[FECM] <b><i>Eqn. 28-2</i></b>

For FEC1: RMII_MODE = RCR1[MII_MODE] && MISCCR[FECM] <b><i>Eqn. 28-3</i></b>

0 FEC configured for MII or 7-wire mode as indicated by the MII_MODE bit.
1 FEC configured for RMII operation, only if the MII_MODE bit is set.

To summarize the various settings see the below table.

<table>
<tr><th colspan="2">Function</th><th rowspan="2">MISCCR[FECM]</th><th rowspan="2">MII_MODE</th><th rowspan="2">RMII_MODE
(Equation 28-2)</th></tr>
<tr><th>FEC0</th><th>FEC1</th></tr>
<tr><td>RMII</td><td>RMII</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>7-wire mode</td><td>—</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>MII mode</td><td>—</td><td>0</td><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>7–6</td><td>Reserved, must be cleared.</td></tr>
<tr><td>5
FCE</td><td>Flow control enable. If asserted, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.</td></tr>
<tr><td>4
BC_REJ</td><td>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.</td></tr>
<tr><td>3
PROM</td><td>Promiscuous mode. All frames are accepted regardless of address matching.</td></tr>
<tr><td>2
MII_MODE</td><td>Media independent interface mode. Selects the external interface mode for transmit and receive blocks.
0 7-wire mode (used only for serial 10 Mbps)
1 MII or RMII mode as indicated by the RMII_MODE bit</td></tr>
<tr><td>1
DRT</td><td>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).</td></tr>
<tr><td>0
LOOP</td><td>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<i>n</i>_TXCLK when LOOP is asserted. DRT must be set to 0 when setting LOOP.</td></tr>
</table>

28.4.11 Transmit Control Registers (TCR0 & TCR1)

TCR*n* is read/write and configures the transmit block. This register is cleared at system reset. Bits 2 and 1 must be modified only when ECR*n*[ETHER_EN] is cleared.

Address: 0xFC03_00C4 (TCR0)
0xFC03_40C4 (TCR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RFC_ PAUSE	TFC_ PAUSE	FDEN	HBC	GTS
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-11. Transmit Control Register (TCR*n*)

Table 28-15. TCR Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 RFC_PAUSE	Receive frame control pause. This read-only status bit is asserted when a full duplex flow control pause frame is received and the transmitter pauses for the duration defined in this pause frame. This bit automatically clears when the pause duration is complete.
3 TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC stops transmission of data frames after the current transmission is complete. At this time, GRA interrupt in the EIR*n* register is asserted. With transmission of data frames stopped, MAC transmits a MAC Control PAUSE frame. Next, the MAC clears the TFC_PAUSE bit and resumes transmitting data frames. If the transmitter pauses due to user assertion of GTS or reception of a PAUSE frame, the MAC may continue transmitting a MAC Control PAUSE frame.
2 FDEN	Full duplex enable. If set, frames transmit independent of carrier sense and collision inputs. This bit should only be modified when ECR*n*[ETHER_EN] is cleared.
1 HBC	Heartbeat control. If set, the heartbeat check performs following end of transmission and the HB bit in the status register is set if the collision input does not assert within the heartbeat window. This bit should only be modified when ECR*n*[ETHER_EN] is cleared.
0 GTS	Graceful transmit stop. When this bit is set, MAC stops transmission after any frame currently transmitted is complete and GRA interrupt in the EIR*n* register is asserted. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. After transmission finishes, clear GTS to restart. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS is set, transmission stops after the collision. The frame is transmitted again after GTS is cleared. There may be old frames in the transmit FIFO that transmit when GTS is reasserted. To avoid this, clear ECR*n*[ETHER_EN] following the GRA interrupt.

28.4.12 Physical Address Lower Registers (PALR0 & PALR1)

PALR*n* contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the 6-byte source address field when transmitting PAUSE frames. This register is not reset and you must initialize it.

Address: 0xFC03_00E4 (PALR0)
0xFC03_40E4 (PALR1)

Access: User read/write

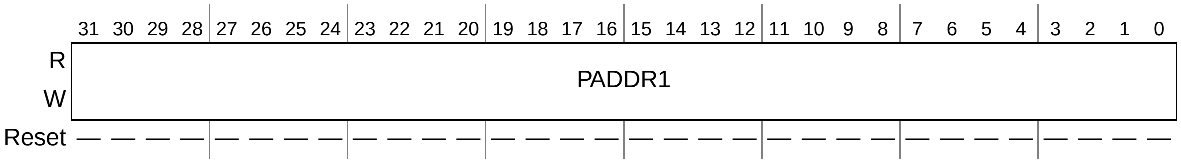


Figure 28-12. Physical Address Lower Register (PALR*n*)

Table 28-16. PALR*n* Field Descriptions

Field	Description
31–0 PADDR1	Bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8), and 3 (bits 7:0) of the 6-byte individual address are used for exact match and the source address field in PAUSE frames.

28.4.13 Physical Address Upper Registers (PAUR0 & PAUR1)

PAUR*n* contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the 6-byte Source Address field when transmitting PAUSE frames. Bits 15:0 of PAUR*n* contain a constant type field (0x8808) for transmission of PAUSE frames. The upper 16 bits of this register are not reset and you must initialize it.

Address: 0xFC03_00E8 (PAUR0)
0xFC03_40E8 (PAUR1)

Access: User read/write

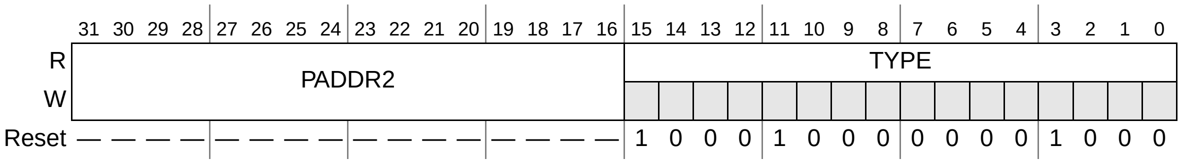


Figure 28-13. Physical Address Upper Register (PAUR*n*)

Table 28-17. PAUR*n* Field Descriptions

Field	Description
31–16 PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address used for exact match, and the source address field in PAUSE frames.
15–0 TYPE	Type field in PAUSE frames. These 16 read-only bits are a constant value of 0x8808.

28.4.14 Opcode/Pause Duration Registers (OPD0 & OPD1)

The OPD*n* is read/write accessible. This register contains the 16-bit opcode and 16-bit pause duration fields used in transmission of a PAUSE frame. The opcode field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration field. The lower 16 bits of this register are not reset and you must initialize them.

Address: 0xFC03_00EC (OPD0)
0xFC03_40EC (OPD1)

Access: User read/write

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

Figure 28-14. Opcode/Pause Duration Register (OPD*n*)

Table 28-18. OPD*n* Field Descriptions

Field	Description
31–16 OPCODE	Opcode field used in PAUSE frames. These read-only bits are a constant, 0x0001.
15–0 PAUSE_DUR	Pause Duration field used in PAUSE frames.

28.4.15 Descriptor Individual Upper Address Registers (IAUR0 & IAUR1)

IAUR*n* contains the upper 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the destination address (DA) field of receive frames with an individual DA. This register is not reset and you must initialize it.

Address: 0xFC03_0118 (IAUR0)
0xFC03_4118 (IAUR1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IADDR1																															
W																																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 28-15. Descriptor Individual Upper Address Register (IAUR*n*)

Table 28-19. IAUR*n* Field Descriptions

Field	Description
31–0 IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

28.4.16 Descriptor Individual Lower Address Registers (IALR0 & IALR1)

IALR*n* contains the lower 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the DA field of receive frames with an individual DA. This register is not reset and you must initialize it.

Address: 0xFC03_011C (IALR0)
0xFC03_411C (IALR1)

Access: User read/write

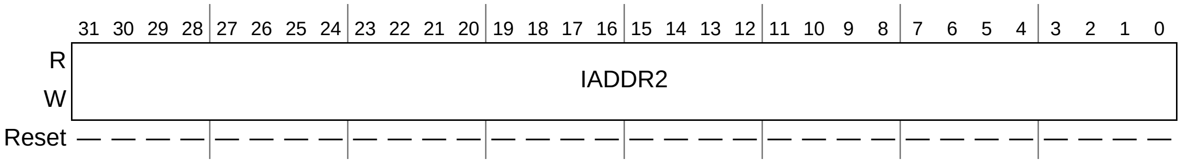


Figure 28-16. Descriptor Individual Lower Address Register (IALR*n*)

Table 28-20. IALR*n* Field Descriptions

Field	Description
31–0 IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0.

28.4.17 Descriptor Group Upper Address Registers (GAUR0 & GAUR1)

GAUR*n* contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

Address: 0xFC03_0120 (GAUR0)
0xFC03_4120 (GAUR1)

Access: User read/write

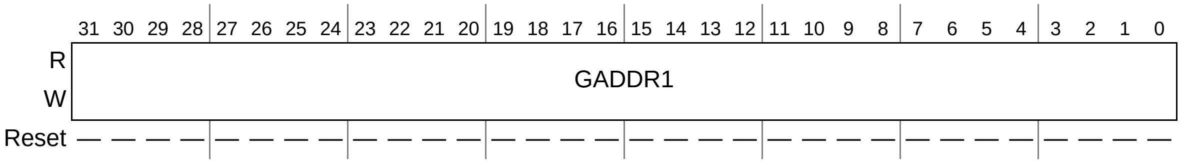


Figure 28-17. Descriptor Group Upper Address Register (GAUR*n*)

Table 28-21. GAUR*n* Field Descriptions

Field	Description
31–0 GADDR1	The GADDR1 register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

28.4.18 Descriptor Group Lower Address Registers (GALR0 & GALR1)

GALR*n* contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

Address: 0xFC03_0124 (GALR0)
0xFC03_4124 (GALR1)

Access: User read/write

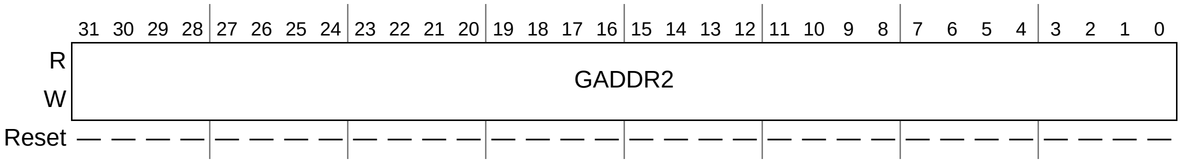


Figure 28-18. Descriptor Group Lower Address Register (GALR*n*)

Table 28-22. GALR*n* Field Descriptions

Field	Description
31–0 GADDR2	The GADDR2 register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

28.4.19 Transmit FIFO Watermark Registers (TFWR0 & TFWR1)

The TFWR*n* controls the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows you to minimize transmit latency (TFWR = 00 or 01) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

Address: 0xFC03_0144 (TFWR0)
0xFC03_4144 (TFWR1)

Access: User read/write

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

Figure 28-19. Transmit FIFO Watermark Register (TFWR*n*)

Table 28-23. TFWR*n* Field Descriptions

Field	Description
31–2	Reserved, must be cleared.
1–0 TFWR	Number of bytes written to transmit FIFO before transmission of a frame begins 00 64 bytes written 01 64 bytes written 10 128 bytes written 11 192 bytes written

28.4.20 FIFO Receive Bound Registers (FRBR0 & FRBR1)

FRBR*n* indicates the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR*n*, to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Address: 0xFC03_014C (FRBR0)
0xFC03_414C (FRBR1)

Access: User read-only

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

Figure 28-20. FIFO Receive Bound Register (FRBR*n*)

Table 28-24. FRBR*n* Field Descriptions

Field	Description
31–10	Reserved, read as 0 (except bit 10, which is read as 1).
9–2 R_BOUND	Read-only. Highest valid FIFO RAM address.
1–0	Reserved, read as 0.

28.4.21 FIFO Receive Start Registers (FRSR0 & FRSR1)

FRSR*n* indicates the starting address of the receive FIFO. FRSR*n* marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR*n*. The receive FIFO uses addresses from FRSR*n* to FRBR*n* inclusive.

Hardware initializes the FRSR*n* register at reset. FRSR*n* only needs to be written to change the default value.

Address: 0xFC03_0150 (FRSR0)
0xFC03_4150 (FRSR1)

Access: User read/write

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

Figure 28-21. FIFO Receive Start Register (FRSR*n*)

Table 28-25. FRSR*n* Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10	Reserved, must be set.
9–2 R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs. For proper operation, ensure that R_FSTART is set to 0x48 or greater.
1–0	Reserved, must be cleared.

28.4.22 Receive Descriptor Ring Start Registers (ERDSR0 & ERDSR1)

ERDSR*n* points to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized prior to operation.

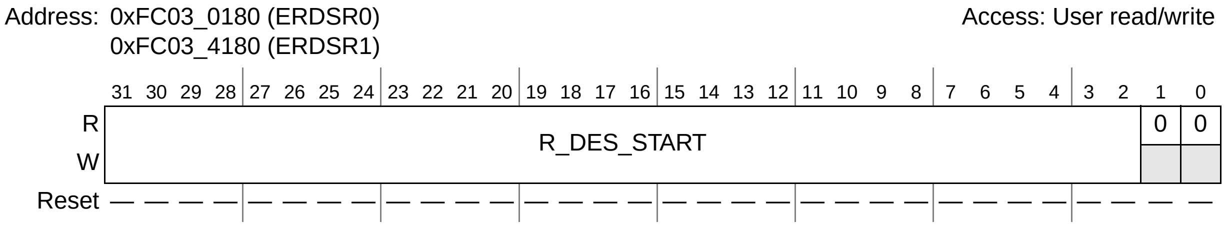


Figure 28-22. Ethernet Receive Descriptor Ring Start Register (ERDSR*n*)

Table 28-26. ERDSR*n* Field Descriptions

Field	Description
31–2 R_DES_START	Pointer to start of receive buffer descriptor queue.
1–0	Reserved, must be cleared.

28.4.23 Transmit Buffer Descriptor Ring Start Registers (ETSDR0 & ETSDR1)

ETSDR*n* provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). You should write zeros to bits 1 and 0. Hardware ignores non-zero values in these two bit positions.

This register is undefined at reset and must be initialized prior to operation.

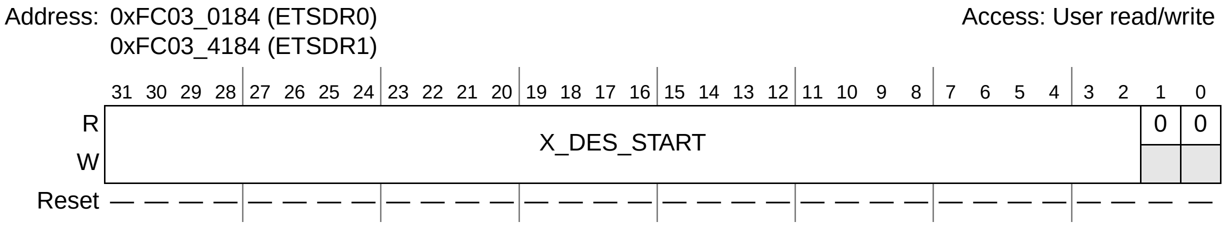


Figure 28-23. Transmit Buffer Descriptor Ring Start Register (ETDSR*n*)

Table 28-27. ETDSR*n* Field Descriptions

Field	Description
31–2 X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, must be cleared.

28.4.24 Receive Buffer Size Registers (EMRBR0 & EMRBR1)

The EMRBR*n* is a user-programmable register that dictates the maximum size of all receive buffers. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, EMRBR*n* must be set to RCR*n*[MAX_FL] or larger. To properly align the buffer, EMRBR*n* must be evenly divisible by 16. To ensure this, bits 3–0 are forced low.

To minimize bus utilization (descriptor fetches), it is recommended that EMRBR*n* be greater than or equal to 256 bytes.

The EMRBR*n* register is undefined at reset and must be initialized by the user.

Address: 0xFC03_0188 (EMRBR0)
0xFC03_4188 (EMRBR1)

Access: User read/write

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

Figure 28-24. Receive Buffer Size Register (EMRBR*n*)

Table 28-28. EMRBR*n* Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10–4 R_BUF_SIZE	Maximum size of receive buffer size in bytes. To minimize bus utilization (descriptor fetches), set this field to 256 bytes (0x10) or larger. 0x10 256 + 15 bytes (minimum size recommended) 0x11 272 + 15 bytes ... 0x7F 2032 + 15 bytes. The FEC writes up to 2047 bytes in the receive buffer. If data larger than 2047 is received, the FEC truncates it and shows 0x7FF in the receive descriptor
3–0	Reserved, must be cleared.

28.5 Functional Description

This section describes the operation of the FECs, beginning with the buffer descriptors, the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FECs.

28.5.1 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.

28.5.1.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames resides in one or more memory buffers external to the FEC. Associated with each buffer is a buffer descriptor (BD), which contains a starting address (32-bit aligned pointer), data length, and status/control information (which contains the current state for the buffer). To permit maximum user flexibility, the BDs are also located in external memory and are read by the FEC DMA engine.

Software produces buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD*n*[E] or TxBD*n*[R] bit produces the buffer. Software writing to TDAR*n* or RDAR*n* tells the FEC that a buffer is placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and consumes the buffers after they have been produced. After the data DMA is complete and the DMA engine writes the buffer descriptor status bits, hardware clears RxBD*n*[E] or TxBD*n*[R] to signal the buffer has been consumed. Software may poll the BDs to detect when the buffers are consumed or may rely on the buffer/frame interrupts. The driver may process these buffers, and they can return to the free list.

The ECR*n*[ETHER_EN] bit operates as a reset to the BD/DMA logic. When ECR*n*[ETHER_EN] is cleared, the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before ECR*n*[ETHER_EN] is set.

The buffer descriptors operate as two separate rings. ERDSR*n* defines the starting address for receive BDs and ETDSR*n* defines the starting address for transmit BDs. The wrap (W) bit defines the last buffer descriptor in each ring. When W is set, the next descriptor in the ring is at the location pointed to by ERDSR*n* and ETDSR*n* for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

28.5.1.1.1 Driver/DMA Operation with Transmit BDs

Typically, a transmit frame is divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a second buffer, IP header in a third buffer, and Ethernet/IEEE 802.3 header in a fouth buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type field(s)), so the driver must provide this in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. TxBD*n*[TC], which must be set by the driver, determines whether the MAC or driver appends the CRC.

The driver (TxBD software producer) should set up Tx BDs so a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length, and control (W, L, TC, ABC) and then the TxBD*n*[R] bit should be set in reverse order (third, second, then first BD) to ensure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA controller could DMA the first BD before the second was made available, potentially causing a transmit FIFO underrun.

In the FEC, the driver notifies the DMA that new transmit frame(s) are available by writing to TDAR*n*. When this register is written to (data value is not significant) the FEC, RISC tells the DMA to read the next transmit BD in the ring. After started, the RISC + DMA continues to read and interpret transmit BDs in order and DMA the associated buffers until a transmit BD is encountered with the R bit cleared. At this point, the FEC polls this BD one more time. If the R bit is cleared the second time, RISC stops the transmit descriptor read process until software sets up another transmit frame and writes to TDAR*n*.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

28.5.1.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore, the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR*n* register.

The driver (RxBD software producer) should set up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) consumes these buffers by filling them with data as frames are received and clearing the E bit and writing to the L bit (1 indicates last buffer in frame), the frame status bits (if L is set), and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD is written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end-of-frame buffers, the receive BD is written with L set and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer is written with the length of the entire frame, not only the length of the last buffer.

For simplicity, the driver may assign a large enough default receive buffer length to contain an entire frame, keeping in mind that a malfunction on the network or out-of-spec implementation could result in giant frames. Frames of 2K (2048) bytes or larger are truncated by the FEC at 2047 bytes so software never sees a receive frame larger than 2047 bytes.

Similar to transmit, the FEC polls the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR*n* register. As frames are received, the FEC fills receive buffers and updates the associated BDs, then reads the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit cleared, it polls this BD once more. If RxBD*n*[E] is clear a second time, FEC stops reading receive BDs until the driver writes to RDAR*n*.

28.5.1.2 Ethernet Receive Buffer Descriptors (RxBD0 & RxBD1)

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in the second longword. When the buffer has been DMA'd, the Ethernet controller modifies the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and writes the length of the used portion of the buffer in the first longword. The M, BC, MC, LG, NO, CR, OV, and TR bits in the first longword of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2	Data Length															
Offset + 4	Rx Data Buffer Pointer - A[31:16]															
Offset + 6	Rx Data Buffer Pointer - A[15:0]															

Figure 28-25. Receive Buffer Descriptor (RxBD*n*)

Table 28-29. Receive Buffer Descriptor Field Definitions

Word	Field	Description
Offset + 0	15 E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD is filled with received data, or data reception has aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	14 RO1	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
Offset + 0	13 W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ERDSR.*n*
Offset + 0	12 RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
Offset + 0	11 L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	10–9	Reserved, must be cleared.
Offset + 0	8 M	Miss. Written by the FEC. This bit is set by the FEC for frames accepted in promiscuous mode, but flagged as a miss by the internal address recognition. Therefore, while in promiscuous mode, you can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is set. 0 The frame was received because of an address recognition hit. 1 The frame was received because of promiscuous mode.
Offset + 0	7 BC	Set if the DA is broadcast (FFFF_FFFF_FFFF).
Offset + 0	6 MC	Set if the DA is multicast and not BC.
Offset + 0	5 LG	Rx frame length violation. Written by the FEC. A frame length greater than RCR*n*[MAX_FL] was recognized. This bit is valid only if the L-bit is set. The receive data is not altered in any way unless the length exceeds 2047 bytes.
Offset + 0	4 NO	Receive non-octet aligned frame. Written by the FEC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error. This bit is valid only if the L-bit is set. If this bit is set, the CR bit is not set.
Offset + 0	3	Reserved, must be cleared.
Offset + 0	2 CR	Receive CRC error. Written by the FEC. This frame contains a CRC error and is an integral number of octets in length. This bit is valid only if the L-bit is set.
Offset + 0	1 OV	Overrun. Written by the FEC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and are zero. This bit is valid only if the L-bit is set.
Offset + 0	0 TR	Set if the receive frame is truncated (frame length > 2047 bytes). If the TR bit is set, the frame must be discarded and the other error bits must be ignored as they may be incorrect.
Offset + 2	15–0 Data Length	Data length. Written by the FEC. Data length is the number of octets written by the FEC into this BD's data buffer if L equals 0 (the value is equal to EMRBR), or the length of the frame including CRC if L is set. It is written by the FEC once as the BD is closed.

Table 28-29. Receive Buffer Descriptor Field Definitions (continued)

Word	Field	Description
0ffset + 4	15–0 A[31:16]	RX data buffer pointer, bits [31:16][1]
Offset + 6	15–0 A[15:0]	RX data buffer pointer, bits [15:0]

[1] The receive buffer pointer, containing the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. The Ethernet controller never modifies this value.

NOTE

When the software driver sets an E bit in one or more receive descriptors, the driver should follow with a write to RDAR.*n*

28.5.1.3 Ethernet Transmit Buffer Descriptors (TxBD0 & TxBD1)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel’s TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (TxBD*n*[R]) when DMA of the buffer is complete. In the TxBD, the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword and the buffer pointer in the second longword.

The FEC clears the R bit when the buffer is transferred. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See Section 28.4.1, “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 28-26. Transmit Buffer Descriptor (TxBD*n*)

Table 28-30. 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.

Table 28-30. Transmit Buffer Descriptor Field Definitions (continued)

Word	Field	Description
Offset + 0	13 W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ETDSR.*n*
Offset + 0	12 TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	11 L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame 1 The buffer is the last in the transmit frame
Offset + 0	10 TC	Transmit CRC. Written by user (only valid if L is set). 0 End transmission immediately after the last data byte 1 Transmit the CRC sequence after the last data byte
Offset + 0	9 ABC	Append bad CRC. Written by user (only valid if L is set). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value)
Offset + 0	8–0	Reserved, must be cleared.
Offset + 2	15–0 Data Length	Data length, written by user. Data length is the number of octets the FEC should transmit from this BD’s data buffer. It is never modified by the FEC.
Offset + 4	15–0 A[31:16]	Tx data buffer pointer, bits [31:16][1]
Offset + 6	15–0 A[15:0]	Tx data buffer pointer, bits [15:0]

[1] The transmit buffer pointer, containing the address of the associated data buffer, must always be evenly divisible by 4. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

After the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame is setting the R bit in the first BD for the frame. The driver must follow that with a write to TDAR*n* that triggers the FEC to poll the next BD in the ring.

28.5.2 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations you must initialize prior to enabling the FECs.

28.5.2.1 Hardware Controlled Initialization

In the FEC, hardware resets registers and control logic that generate interrupts. A hardware reset negates output signals and resets general configuration bits.

Other registers reset when the ECR*n*[ETHER_EN] bit is cleared (which is accomplished by a hard reset or software to halt operation). By clearing ECR*n*[ETHER_EN], configuration control registers such as the TCR*n* and RCR*n* are not reset, but the entire data path is reset.

Table 28-31. ECR*n*[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR*n*	Cleared
TDAR*n*	Cleared
Descriptor Controller block	Halt operation

28.5.3 User Initialization (Prior to Setting ECR*n*[ETHER_EN])

You need to initialize portions the FEC prior to setting the ECR*n*[ETHER_EN] bit. The exact values depend on the particular application. The sequence is not important.

Table 28-32 defines Ethernet MAC registers requiring initialization.

Table 28-32. User Initialization (Before ECR*n*[ETHER_EN])

Description
Initialize EIMR*n*
Clear EIR*n* (write 0xFFFF_FFFF)
TFWR*n* (optional)
IALR*n* / IAUR*n*
GAUR*n* / GALR*n*
PALR*n* / PAUR*n* (only needed for full duplex flow control)
OPD*n* (only needed for full duplex flow control)
RCR*n*
TCR*n*
MSCR*n* (optional)
Clear MIB_RAM*n*

Table 28-33 defines FEC FIFO/DMA registers that require initialization.

Table 28-33. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR*n* (optional)
Initialize EMRBR*n*
Initialize ERDSR*n*
Initialize ETDSR*n*

Table 28-33. FEC User Initialization (Before ECR[ETHER_EN]) (continued)

Description
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

28.5.4 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR*n*[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, hardware is ready for operation.

Table 28-34 shows microcontroller initialization operations.

Table 28-34. 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

28.5.5 User Initialization (After Setting ECR*n*[ETHER_EN])

After setting ECR*n*[ETHER_EN], you can set up the buffer/frame descriptors and write to TDAR*n* and RDAR*n*. Refer to Section 28.5.1, “Buffer Descriptors,” for more details.

28.5.6 Network Interface Options

The FECs support an MII and reduced MII interface for 10/100 Mbps Ethernet, as well as a 7-wire serial interface for 10 Mbps Ethernet. The RCR*n*[MII_MODE] and RCR*n*[RMII_MODE] bits selects the interface mode. In MII mode (RCR*n*[MII_MODE] set and RCR*n*[RMII_MODE] cleared), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. Table 28-35 shows these signals.

Table 28-35. MII Mode

Signal Description	EMAC pin
Transmit Clock	FEC*n*_TXCLK
Transmit Enable	FEC*n*_TXEN
Transmit Data	FEC*n*_TXD[3:0]
Transmit Error	FEC*n*_TXER

Table 28-35. MII Mode (continued)

Signal Description	EMAC pin
Collision	FEC*n*_COL
Carrier Sense	FEC*n*_CRS
Receive Clock	FEC*n*_RXCLK
Receive Data Valid	FEC*n*_RXDV
Receive Data	FEC*n*_RXD[3:0]
Receive Error	FEC*n*_RXER
Management Data Clock	FEC*n*_MDC
Management Data Input/Output	FEC*n*_MDIO

In RMII mode (RCR*n*[MII_MODE] set and RCR*n*[RMII_MODE] cleared), the EMAC supports 8 external signals. These signals are shown in Table 28-36 below.

Table 28-36. RMII Mode Configuration

Signal Description	EMAC Pin
Reference Clock	FEC*n*_TXCLK
Transmit Enable	FEC*n*_TXEN
Transmit Data	FEC*n*_TXD[1:0]
Receive Data Valid	FEC*n*_RXDV
Receive Data	FEC*n*_RXD[1:0]
Receive Error	FEC*n*_RXER

The 7-wire serial mode interface (RCR*n*[MII_MODE] cleared and RCR*n*[RMII_MODE] cleared or set) is generally referred to as AMD mode. Table 28-37 shows the 7-wire mode connections to the external transceiver.

Table 28-37. 7-Wire Mode Configuration

Signal description	EMAC Pin
Transmit Clock	FEC*n*_TXCLK
Transmit Enable	FEC*n*_TXEN
Transmit Data	FEC*n*_TXD[0]
Collision	FEC*n*_COL
Receive Clock	FEC*n*_RXCLK

Table 28-37. 7-Wire Mode Configuration (continued)

Signal description	EMAC Pin
Receive Data Valid	FEC*n*_RXDV
Receive Data	FEC*n*_RXD[0]

28.5.7 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. After ECR*n*[ETHER_EN] is set and data appears in the transmit FIFO, the Ethernet MAC can transmit onto the network. The Ethernet controller transmits bytes least significant bit (lsb) first.

When the transmit FIFO fills to the watermark (defined by TFWR*n*), MAC transmit logic asserts FEC*n*_TXEN and starts transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (FEC*n*_CRS is asserted). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See Section 28.5.17.1, “Transmission Errors,” for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data is transmitted, FCS (frame check sequence) or 32-bit cyclic redundancy check (CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC is appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Transmit logic automatically pads short frames (if the TC bit in the transmit buffer descriptor for the end of frame buffer is set).

Settings in the EIMR*n* determine interrupts generated to the buffer (TXB) and frame (TFINT).

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL bytes, BABT interrupt is asserted. However, the entire frame is transmitted (no truncation).

To pause transmission, set TCR*n*[GTS] (graceful transmit stop). The FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame finishes or terminates with a collision. After the transmitter has stopped, the GRA (graceful stop complete) interrupt is asserted. If TCR*n*[GTS] is cleared, the FEC resumes transmission with the next frame.

28.5.7.1 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.

28.5.8 FEC Frame Reception

The FEC receivers work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking. The Ethernet controller receives serial data lsb first.

When the driver enables the FEC receiver by setting ECR*n*[ETHER_EN], it immediately starts processing receive frames. When FEC*n*_RXDV is asserted, the receiver first checks for a valid PA/SFD header. If the PA/SFD is valid, it is stripped and the receiver processes the frame. If a valid PA/SFD is not found, the frame is ignored.

In serial mode, the first 16 bit times of RX_D0 following assertion of FEC*n*_RXDV are ignored. Following the first 16 bit times, the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame are received, the FEC performs address recognition on the frame.

After a collision window (64 bytes) of data is received and if address recognition has not rejected the frame, the receive FIFO signals the frame is accepted and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to reject the frame. Therefore, no collision fragments are presented to you except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and after the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV, and TR status bits, and the frame length. See Section 28.5.17.2, “Reception Errors,” for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR*n* register. A receive error interrupt is a babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt occurs and the LG bit in the receive buffer descriptor (RxBD*n*) is set. See Section 28.5.1.2, “Ethernet Receive Buffer Descriptors (RxBD0 & RxBD1),” for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD*n*, writes the other frame status bits into the RxBD*n*, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR*n*, maskable by RFIEN bit in EIMR*n*), indicating that a frame is received and is in memory. The Ethernet controller then waits for a new frame.

28.5.9 Ethernet Address Recognition

The FECs filter the received frames based on destination address (DA) type — individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames appears in the figures below.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in Figure 28-27 illustrates the address recognition decisions made by the receive block, while Figure 28-28 illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR*n*[BC_REJ]) is cleared, then the frame is accepted unconditionally, as shown in Figure 28-27. Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in Figure 28-28.

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller performs a group hash table lookup using the 64-entry hash table programmed in GAUR*n* and GALR*n*. If a hash match occurs, the receiver accepts the frame.

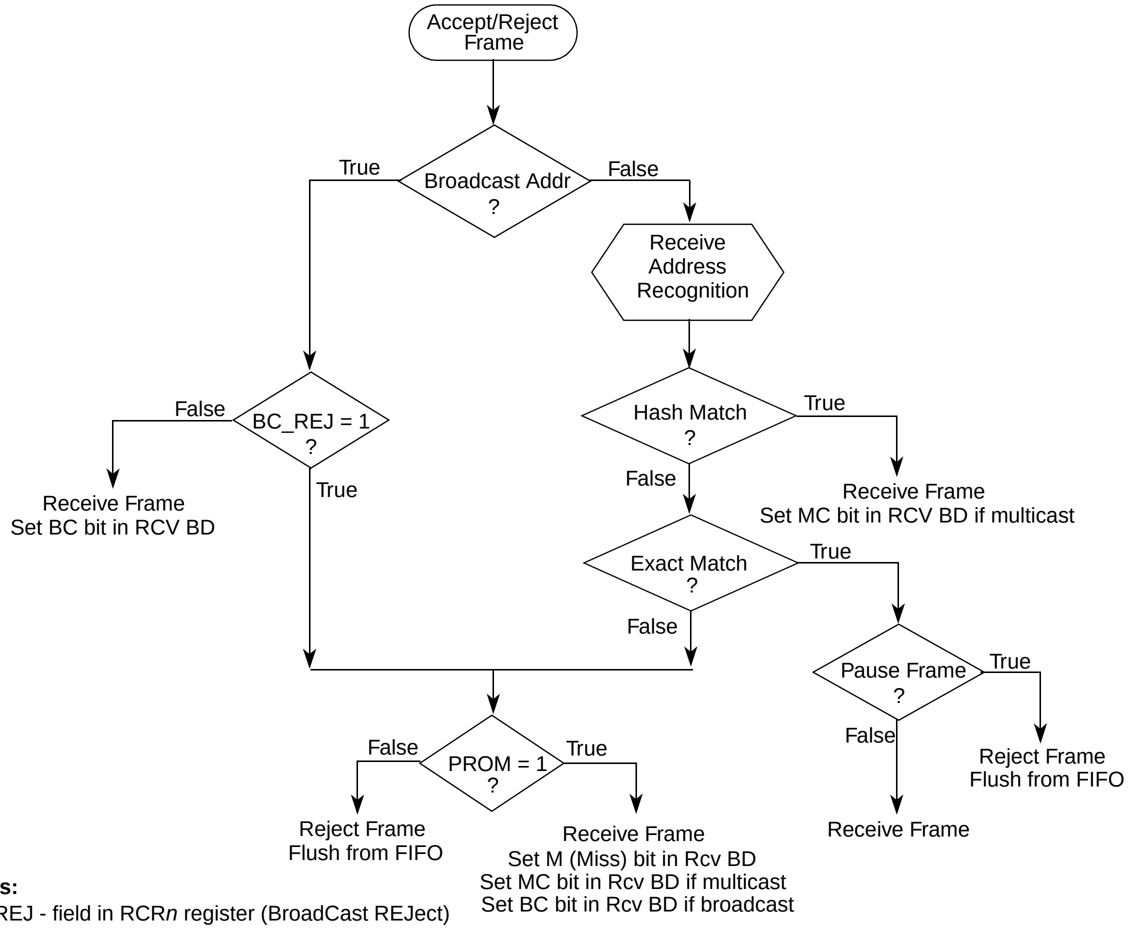
If flow control is enabled, the microcontroller does an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines the received frame is a valid PAUSE frame, the frame is rejected. The receiver detects a PAUSE frame with the DA field set to the designated PAUSE DA or the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that you program in the PALR*n* and PAUR*n* registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR*n* and IALR*n*. In the case of an individual hash match, the frame is accepted. Again, the receiver accepts or rejects the frame based on PAUSE frame detection, shown in Figure 28-27.

If neither a hash match (group or individual) nor an exact match (group or individual) occur, and if promiscuous mode is enabled (RCR*n*[PROM] set), the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

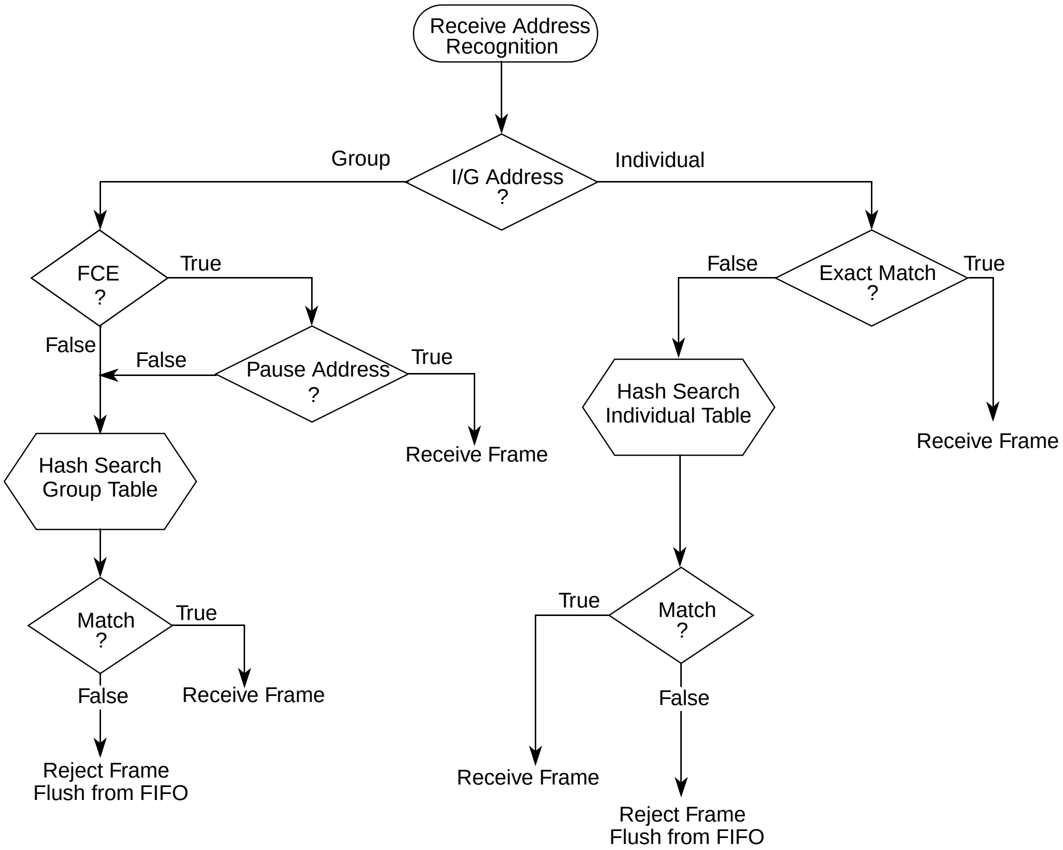
Similarly, if the DA is a broadcast address, broadcast reject (RCR*n*[BC_REJ]) is asserted, and promiscuous mode is enabled, the frame is accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame is rejected.

In general, when a frame is rejected, it is flushed from the FIFO.



Notes:
BC_REJ - field in RCR*n* register (BroadCast REJect)
PROM - field in RCR*n* register (PROMiscous mode)
Pause Frame - valid PAUSE frame received

Figure 28-27. Ethernet Address Recognition—Receive Block Decisions



Notes:
FCE - field in RCR*n* register (flow control enable)
I/G - Individual/Group bit in destination address (lsb in first byte received in MAC frame)

Figure 28-28. Ethernet Address Recognition—Microcode Decisions

28.5.10 Hash Algorithm

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, represented by 64 bits stored in GAUR*n*, GALR*n* (group address hash match), or IAUR*n*, IALR*n* (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the six most significant bits of the CRC-encoded result to generate a number between 0 and 63. The msb of the CRC result selects GAUR*n* (msb = 1) or GALR*n* (msb = 0). The five least significant bits of the hash result select the bit within the selected register. If the CRC generator selects a bit set in the hash table, the frame is accepted; otherwise, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The user must initialize the hash table registers. Use this CRC32 polynomial to compute the hash:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1 \qquad \textit{Eqn. 28-4}$$

Table 28-38 contains example destination addresses and corresponding hash values.

Table 28-38. Destination Address to 6-Bit Hash

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65FF_FFFF_FFFF	0x0	0
55FF_FFFF_FFFF	0x1	1
15FF_FFFF_FFFF	0x2	2
35FF_FFFF_FFFF	0x3	3
B5FF_FFFF_FFFF	0x4	4
95FF_FFFF_FFFF	0x5	5
D5FF_FFFF_FFFF	0x6	6
F5FF_FFFF_FFFF	0x7	7
DBFF_FFFF_FFFF	0x8	8
FBFF_FFFF_FFFF	0x9	9
BBFF_FFFF_FFFF	0xA	10
8BFF_FFFF_FFFF	0xB	11
0BFF_FFFF_FFFF	0xC	12
3BFF_FFFF_FFFF	0xD	13
7BFF_FFFF_FFFF	0xE	14
5BFF_FFFF_FFFF	0xF	15
27FF_FFFF_FFFF	0x10	16
07FF_FFFF_FFFF	0x11	17
57FF_FFFF_FFFF	0x12	18
77FF_FFFF_FFFF	0x13	19
F7FF_FFFF_FFFF	0x14	20
C7FF_FFFF_FFFF	0x15	21
97FF_FFFF_FFFF	0x16	22
A7FF_FFFF_FFFF	0x17	23
99FF_FFFF_FFFF	0x18	24
B9FF_FFFF_FFFF	0x19	25
F9FF_FFFF_FFFF	0x1A	26
C9FF_FFFF_FFFF	0x1B	27

Table 28-38. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
59FF_FFFF_FFFF	0x1C	28
79FF_FFFF_FFFF	0x1D	29
29FF_FFFF_FFFF	0x1E	30
19FF_FFFF_FFFF	0x1F	31
D1FF_FFFF_FFFF	0x20	32
F1FF_FFFF_FFFF	0x21	33
B1FF_FFFF_FFFF	0x22	34
91FF_FFFF_FFFF	0x23	35
11FF_FFFF_FFFF	0x24	36
31FF_FFFF_FFFF	0x25	37
71FF_FFFF_FFFF	0x26	38
51FF_FFFF_FFFF	0x27	39
7FFF_FFFF_FFFF	0x28	40
4FFF_FFFF_FFFF	0x29	41
1FFF_FFFF_FFFF	0x2A	42
3FFF_FFFF_FFFF	0x2B	43
BFFF_FFFF_FFFF	0x2C	44
9FFF_FFFF_FFFF	0x2D	45
DFFF_FFFF_FFFF	0x2E	46
EFFF_FFFF_FFFF	0x2F	47
93FF_FFFF_FFFF	0x30	48
B3FF_FFFF_FFFF	0x31	49
F3FF_FFFF_FFFF	0x32	50
D3FF_FFFF_FFFF	0x33	51
53FF_FFFF_FFFF	0x34	52
73FF_FFFF_FFFF	0x35	53
23FF_FFFF_FFFF	0x36	54
13FF_FFFF_FFFF	0x37	55
3DFF_FFFF_FFFF	0x38	56
0DFF_FFFF_FFFF	0x39	57
5DFF_FFFF_FFFF	0x3A	58
7DFF_FFFF_FFFF	0x3B	59

Table 28-38. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
FDFF_FFFF_FFFF	0x3C	60
DDFF_FFFF_FFFF	0x3D	61
9DFF_FFFF_FFFF	0x3E	62
BDFF_FFFF_FFFF	0x3F	63

28.5.11 Full Duplex Flow Control

Full-duplex flow control allows you to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable PAUSE frame detection, the FEC must operate in full-duplex mode (TCR*n*[FDEN] set) with flow control (RCR*n*[FCE] set). The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in Table 28-39. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 28-39. PAUSE Frame Field Specification

48-bit Destination Address	0x0180_C200_0001 or Physical Address
48-bit Source Address	Any
16-bit Type	0x8808
16-bit Opcode	0x0001
16-bit PAUSE Duration	0x0000 – 0xFFFF

The receiver and microcontroller modules perform PAUSE frame detection. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, TCR*n*[GTS] is set by the FEC internally. When transmission has paused, the EIR*n*[GRA] interrupt is asserted and the pause timer begins to increment. The pause timer uses the transmit backoff timer hardware for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD*n*[PAUSE_DUR] slot times have expired. On OPD*n*[PAUSE_DUR] expiration, TCR*n*[GTS] is cleared allowing MAC data frame transmission to resume. The receive flow control pause status bit (TCR*n*[RFC_PAUSE]) is set while the transmitter pauses due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and you must set flow control pause (TCR*n*[TFC_PAUSE]). After TCR*n*[TFC_PAUSE] is set, the transmitter sets TCR*n*[GTS] internally. When the transmission of data frames stops, the EIR*n*[GRA] (graceful stop complete) interrupt asserts and the pause frame is transmitted. TCR*n*[TFC_PAUSE,GTS] are then cleared internally.

You must specify the desired pause duration in the OPD*n* register.

When the transmitter pauses due to receiver/microcontroller pause frame detection, TCR*n*[TFC_PAUSE] may remain set and cause the transmission of a single pause frame. In this case, the EIR*n*[GRA] interrupt is not asserted.

28.5.12 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.

28.5.13 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.

28.5.14 MII Internal and External Loopback

Internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Internal and external loopback are configured using combinations of the RCR*n*[LOOP, DRT] and TCR*n*[FDEN] bits.

Set FDEN for internal and external loopback.

For internal loopback, set RCR*n*[LOOP] and clear RCR*n*[DRT]. FEC*n*_TXEN and FEC*n*_TXER do not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the transmit and receive blocks use the internal bus clock instead of the clocks from the external transceiver. This causes an increase in the required system bus bandwidth for transmit and receive data being DMA'd to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underruns and receive FIFO overflows.

For external loopback, clear RCR*n*[LOOP] and RCR*n*[DRT], and configure the external transceiver for loopback.

28.5.15 RMII Loopback

The FEC also supports RMII loopback. In this mode, transmit data is sent to the RMII receive logic and out the FEC_TXD[1:0] pins. To enable RMII loopback mode, set RCR*n*[RMII_MODE, RMII_LOOP] and TCR*n*[FDEN] and clear RCR*n*[RMII_ECHO, LOOP].

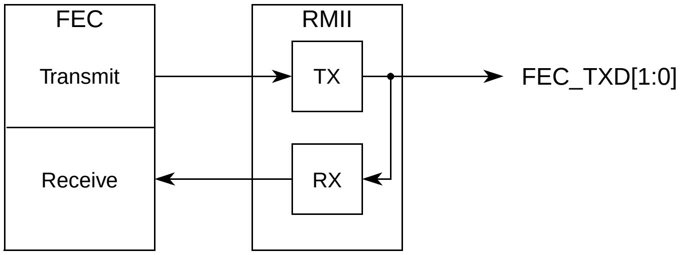


Figure 28-29. RMII Loopback Mode

28.5.16 RMII Echo

The ethernet controller also supports RMII echo mode, which transmits data on FEC_TXD[1:0] as it is received on FEC_RXD[1:0]. To enable RMII echo mode, set RCR*n*[RMII_MODE, RMII_ECHO] bits and clear RCR*n*[RMII_LOOP, LOOP].

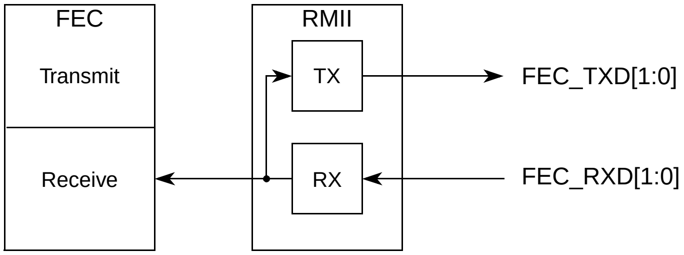


Figure 28-30. RMII Echo Mode

28.5.17 Ethernet Error-Managing Procedure

The Ethernet controller reports frame reception and transmission error conditions using the MIB block counters, the FEC RxBDs, and the EIR*n* register.

28.5.17.1 Transmission Errors

28.5.17.1.1 Transmitter Underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed, and EIR*n*[UN] is set. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame. The UN interrupt is asserted if enabled in the EIMR*n* register.

28.5.17.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and EIR*n*[RL] is set. The FEC then continues to the next transmit buffer descriptor and begins transmitting the next frame. The RL interrupt is asserted if enabled in the EIMR*n* register.

28.5.17.1.3 Late Collision

When a collision occurs after the slot time (512 bits starting at the Preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and EIR*n*[LC] is set. The FEC then continues to the next transmit buffer descriptor and begin transmitting the next frame. The LC interrupt is asserted if enabled in the EIMR*n* register.

28.5.17.1.4 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*n*[HBC] is set and the heartbeat condition is not detected by the FEC after a frame transmission, a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets EIR*n*[HB], and generates the HBERR interrupt if it is enabled.

28.5.17.2 Reception Errors

28.5.17.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, FEC sets RxBD*n*[OV]. All subsequent data in the frame is discarded and subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. The driver must discard this frame.

28.5.17.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller manages up to seven dribbling bits when the receive frame terminates past an non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If there is a CRC error, the frame non-octet aligned (NO) error is reported in the RxBD*n*. If there is no CRC error, no error is reported.

28.5.17.2.3 CRC Error

When a CRC error occurs with no dribble bits, FEC closes the buffer and sets RxBD*n*[CR]. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

28.5.17.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes the BABR interrupt is generated, and RxBD*n*[LG] is set. The frame is not truncated unless the frame length exceeds 2047 bytes.

28.5.17.2.5 Truncation

When the receive frame length exceeds 2047 bytes, frame is truncated and RxBD*n*[TR] is set.

Chapter 29
Synchronous Serial Interface (SSI)

29.1 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 29-1, 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.

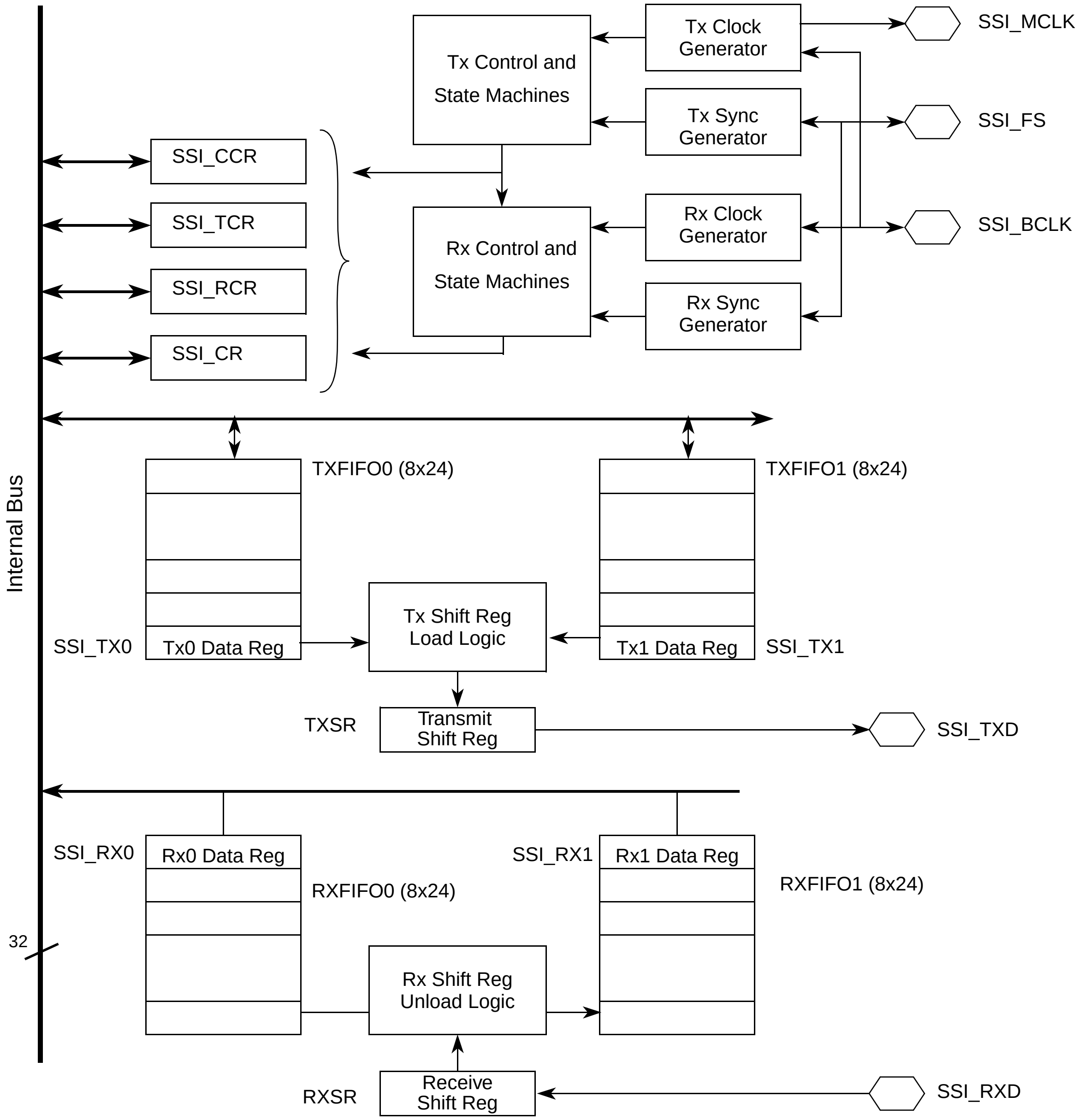


Figure 29-1. SSI Block Diagram

29.1.1 Overview

The SSI is a full-duplex serial port that allows the processor to communicate with a variety of serial devices. Such serial devices are:

- Standard codecs
- Digital signal processors (DSPs)
- Microprocessors
- Peripherals
- Audio codecs that implement the inter-IC sound bus (I^2S) and the Intel® AC97 standards

The SSI module typically transfers samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with shared clock generation and frame synchronization.

NOTE

> The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, "Pin-Multiplexing and Control") prior to configuring the SSI.

29.1.2 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

29.1.3 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 29-1 lists these operating modes and some of the typical applications in which they can be used:

Table 29-1. 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 29.4.1, “Detailed Operating Mode Descriptions,” for more details on the above modes.

29.2 External Signal Description

The five SSI signals are explained below.

Table 29-2. 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

29.2.1 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 29-40.

29.2.2 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.

29.2.3 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.

29.2.4 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.

29.2.5 SSI_RXD — Serial Receive Data

The SSI_RXD port is an input and brings serial data into the receive data shift register.

29.2.6 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 29-2 shows the main SSI configurations. These ports support all transmit and receive functions with continuous or gated clock as shown. Gated clock implementations do not require the use of the frame sync port (SSI_FS).

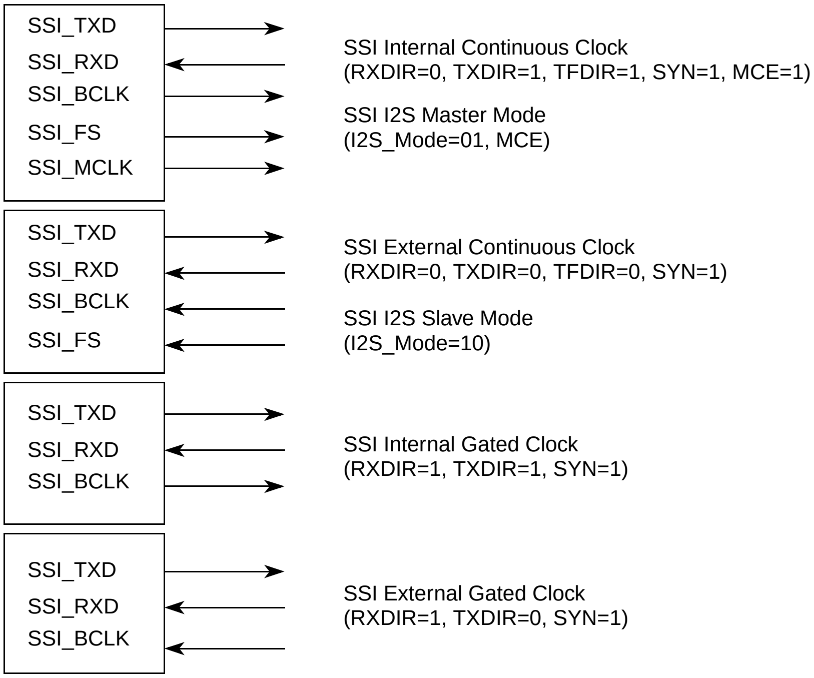


Figure 29-2. Synchronous SSI Configurations—Continuous and Gated Clock

Figure 29-3 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.

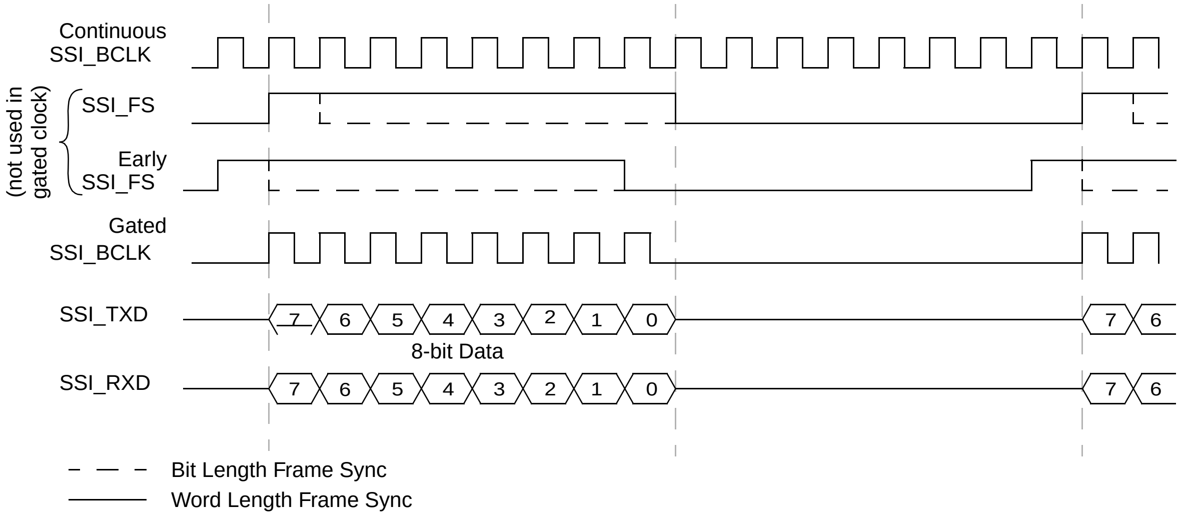


Figure 29-3. Serial Clock and Frame Sync Timing

Table 29-3. 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	—

29.3 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 29-4. 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	29.3.1/29-8
0xFC0B_C004	SSI Transmit Data Register 1 (SSI_TX1)	32	R/W	0x0000_0000	29.3.1/29-8
0xFC0B_C008	SSI Receive Data Register 0 (SSI_RX0)	32	R	0x0000_0000	29.3.4/29-11
0xFC0B_C00C	SSI Receive Data Register 1 (SSI_RX1)	32	R	0x0000_0000	29.3.4/29-11
0xFC0B_C010	SSI Control Register (SSI_CR)	32	R/W	0x0000_0000	29.3.7/29-13
0xFC0B_C014	SSI Interrupt Status Register (SSI_ISR)	32	R/W	0x0000_3003	29.3.8/29-15

Table 29-4. 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	29.3.9/29-21
0xFC0B_C01C	SSI Transmit Configuration Register (SSI_TCR)	32	R/W	0x0000_0200	29.3.10/29-23
0xFC0B_C020	SSI Receive Configuration Register (SSI_RCR)	32	R/W	0x0000_0200	29.3.11/29-24
0xFC0B_C024	SSI Clock Control Register (SSI_CCR)	32	R/W	0x0004_0000	29.3.12/29-25
0xFC0B_C02C	SSI FIFO Control/Status Register (SSI_FCSR)	32	R/W	0x0081_0081	29.3.13/29-27
0xFC0B_C038	SSI AC97 Control Register (SSI_ACR)	32	R/W	0x0000_0000	29.3.14/29-28
0xFC0B_C03C	SSI AC97 Command Address Register (SSI_ACADD)	32	R/W	0x0000_0000	29.3.15/29-29
0xFC0B_C040	SSI AC97 Command Data Register (SSI_ACDAT)	32	R/W	0x0000_0000	29.3.16/29-30
0xFC0B_C044	SSI AC97 Tag Register (SSI_ATAG)	32	R/W	0x0000_0000	29.3.17/29-30
0xFC0B_C048	SSI Transmit Time Slot Mask Register (SSI_TMASK)	32	R/W	0x0000_0000	29.3.18/29-31
0xFC0B_C04C	SSI Receive Time Slot Mask Register (SSI_RMASK)	32	R/W	0x0000_0000	29.3.19/29-31
0xFC0B_C050	SSI AC97 Channel Status Register (SSI_ACCSR)	32	R	0x0000_0000	29.3.20/29-32
0xFC0B_C054	SSI AC97 Channel Enable Register (SSI_ACCEN)	32	R	0x0000_0000	29.3.21/29-32
0xFC0B_C058	SSI AC97 Channel Disable Register (SSI_ACCDIS)	32	R	0x0000_0000	29.3.22/29-33

29.3.1 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 29.4.4, “Supported Data Alignment Formats.”

Address: 0xFC0B_C000 (SSI_TX0)
0xFC0B_C004 (SSI_TX1)

Access: User read/write

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

Figure 29-4. SSI Transmit Data Registers (SSI_TX0, SSI_TX1)

Table 29-5. 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

29.3.2 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.

29.3.3 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.

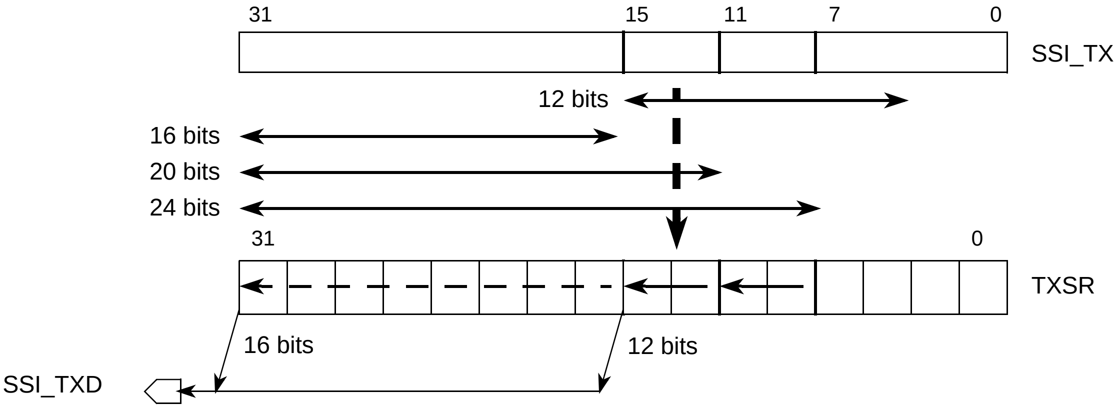


Figure 29-5. Transmit Data Path (TXBIT0=0, TSHFD=0) (msb Alignment)

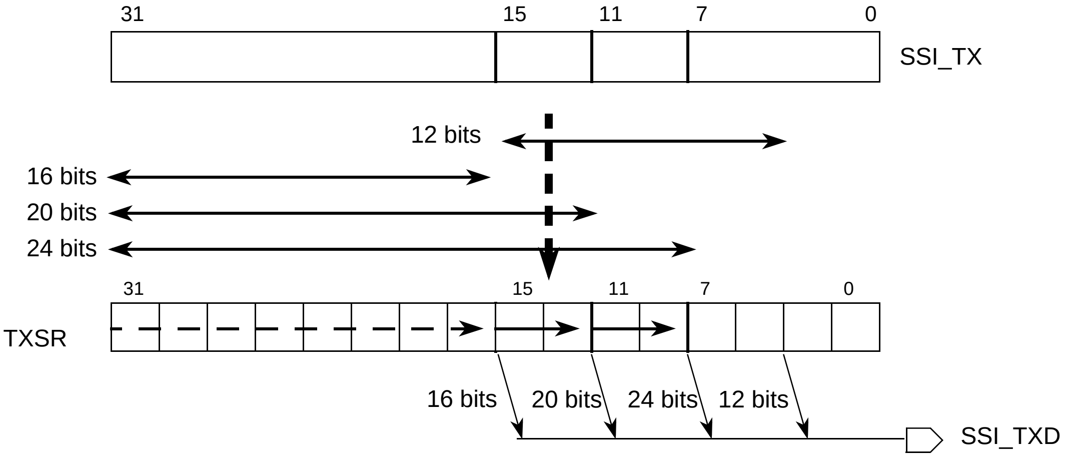


Figure 29-6. Transmit Data Path (TXBIT0=0, TSHFD=1) (msb Alignment)

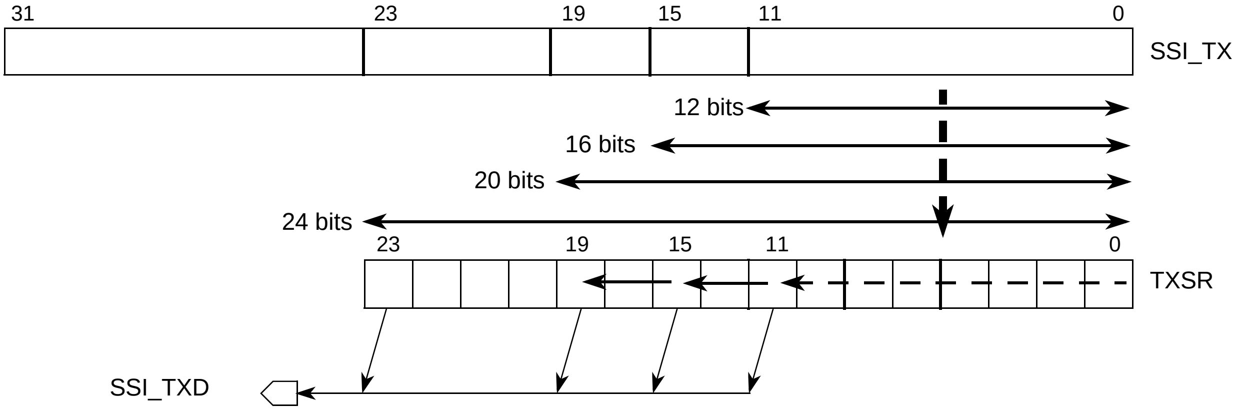


Figure 29-7. Transmit Data Path (TXBIT0=1, TSHFD=0) (lsb Alignment)

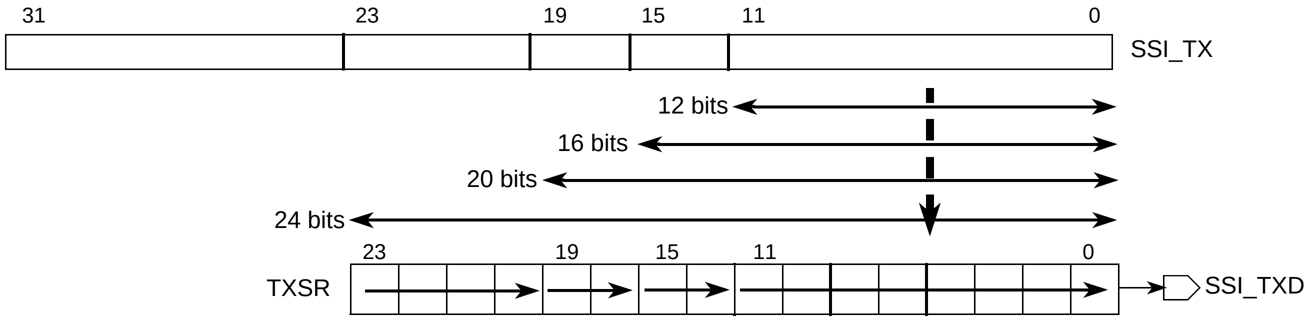


Figure 29-8. Transmit Data Path (TXBIT0=1, TSHFD=1) (lsb Alignment)

29.3.4 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 29.3.6, “SSI Receive Shift Register (RXSR).”

Address: 0xFC0B_C008 (SSI_RX0)
0xFC0B_C00C (SSI_RX1)

Access: User read/write

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

Figure 29-9. SSI Receive Data Registers (SSI_RX0, SSI_RX1)

Table 29-6. 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.

29.3.5 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.

29.3.6 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.

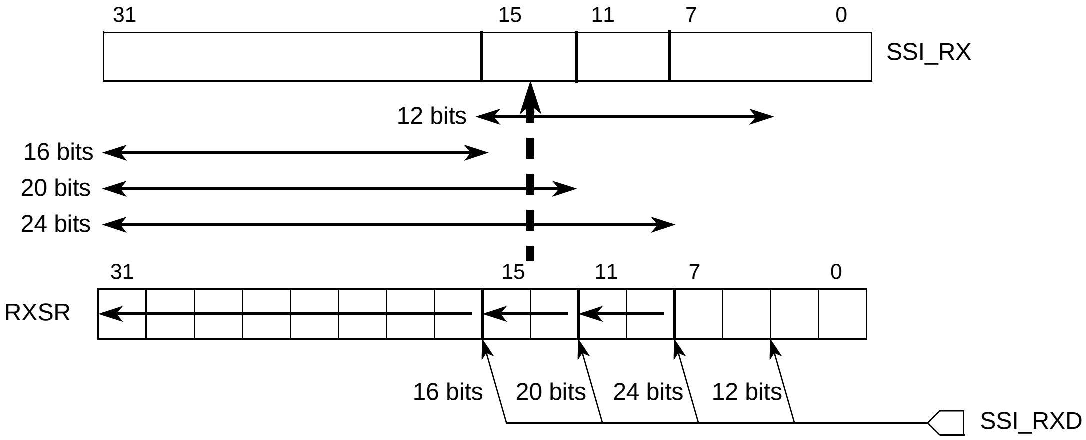


Figure 29-10. Receive Data Path (RXBIT0=0, RSHFD=0) (msb Alignment)

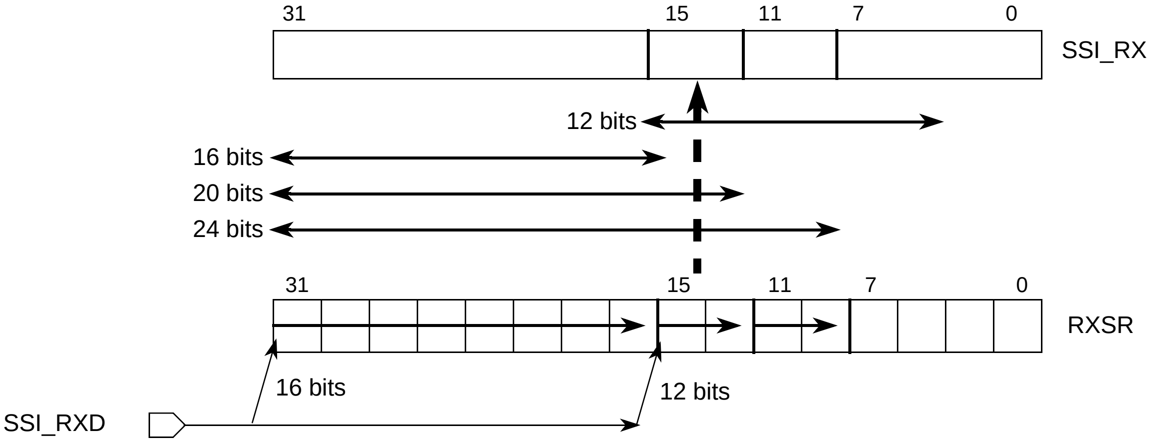


Figure 29-11. Receive Data Path (RXBIT0=0, RSHFD=1) (msb Alignment)

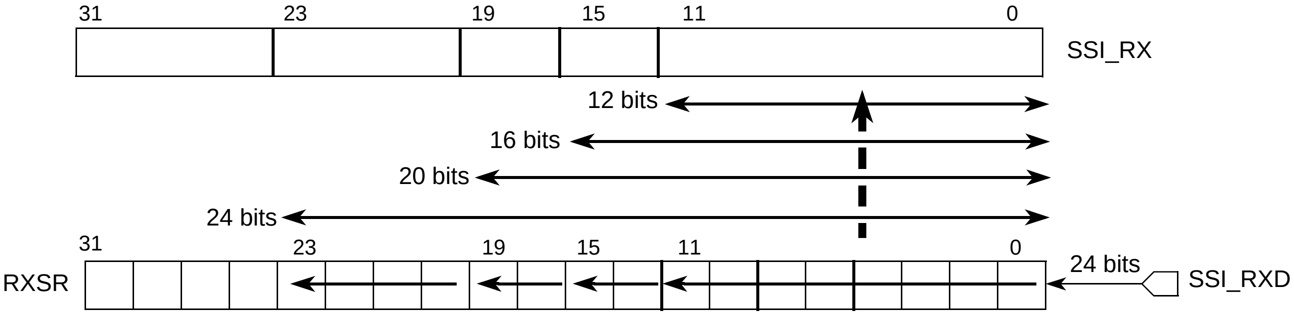


Figure 29-12. Receive Data Path (RXBIT0=1, RSHFD=0) (lsb Alignment)

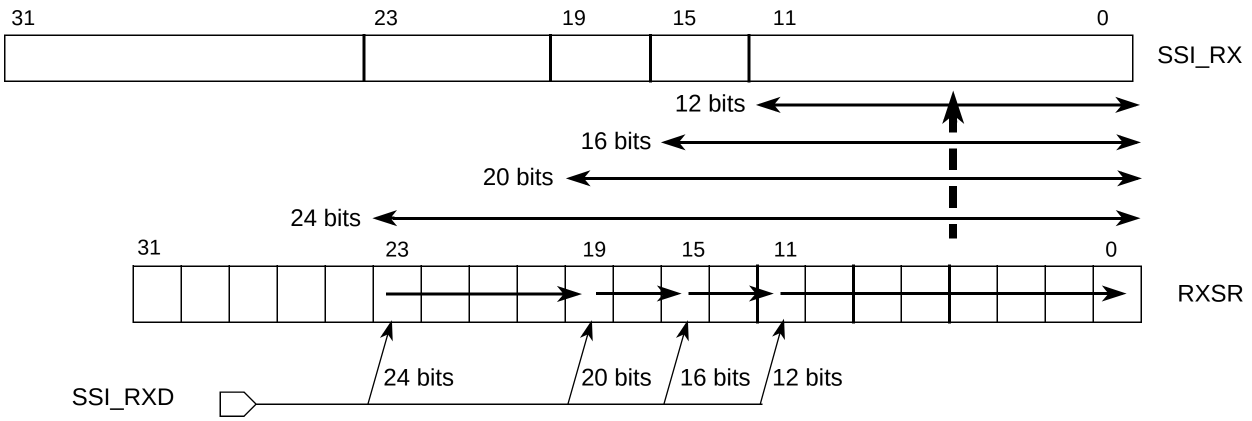


Figure 29-13. Receive Data Path (RXBIT0=1, RSHFD=1) (lsb Alignment)

29.3.7 SSI Control Register (SSI_CR)

The SSI control register sets up the SSI modules. SSI operating modes are selected in this register (except AC97 mode, which is selected in SSI_ACR register).

Address: 0xFC0B_C010 (SSI_CR)   Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	RCD	TCD	CIS	TCH	MCE	I2S		SYN	NET	RE	TE	SSI_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-14. SSI Control Register (SSI_CR)

Table 29-7. SSI_CR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11 RCD	Receive frame clock disable. Disables the frame sync and clock after the current receive frame when the receiver is disabled (RE is cleared). Writing to this bit only has affect when RE is cleared. 0 Continue frame sync and clock generation after the current frame if RE is cleared. Use this setting when the frame sync and clocks are required, even when no data is received. 1 Stop frame sync and clock generation at the next frame boundary if RE is cleared.
10 TCD	Transmit frame clock disable. Disables the frame sync and clock after the current transmit frame when the transmitter is disabled (TE is cleared). Writing to this bit only has affect when TE is cleared. 0 Continue frame sync and clock generation after the current frame if TE is cleared. Use this setting when the frame sync and clocks are required, even when no data is transmitted. 1 Stop frame sync and clock generation at the next frame boundary if TE is cleared.

Table 29-7. SSI_CR Field Descriptions (continued)

Field	Description
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
7 MCE	Master clock enable. Allows the SSI to output the master clock at the SSI_MCLK port, if network mode and transmit internal clock mode are set. The DIV2, PSR, and PM bits determine the relationship between the bit clock (SSI_BCLK) and SSI_MCLK. In I^{2}S master mode, this bit is used to output the oversampling clock on SSI_MCLK. 0 Master clock not output on the SSI_MCLK pin 1 Master clock output on the SSI_MCLK pin
6–5 I2S	I^{2}S mode select. Selects normal, I^{2}S master, or I^{2}S slave mode. Refer to Section 29.4.1.4, “I2S Mode,” for a detailed description of I^{2}S mode. 00 Normal mode 01 I^{2}S master mode 10 I^{2}S slave mode 11 Normal mode
4 SYN	Synchronous mode enable. In synchronous mode, transmit and receive sections of SSI share a common clock port (SSI_BCLK) and frame sync port (SSI_FS). 0 Reserved. 1 Synchronous mode selected.
3 NET	Network mode enable. 0 Network mode not selected 1 Network mode selected
2 RE	Receiver enable. When this bit is set, data reception starts with the arrival of the next frame sync. If data is received when this bit is cleared, data reception continues with the end of the current frame and then stops. If this bit is set again before the second to last bit of the last time slot in the current frame, reception continues without interruption. 0 Receiver disabled 1 Receiver enabled

Table 29-7. SSI_CR Field Descriptions (continued)

Field	Description
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

29.3.8 SSI Interrupt Status Register (SSI_ISR)

The SSI interrupt status register monitors the SSI. This register 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 29.4.5, “Receive Interrupt Enable Bit Description,” and Section 29.4.6, “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 writing one to the corresponding bit in the SSI_ISR.

Address: 0xFC0B_C014 (SSI_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	RFRC	TRFC	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					w1c	w1c	w1c	w1c								
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 29-15. SSI Interrupt Status Register (SSI_ISR)

Table 29-8. SSI_ISR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 RFRC	Receive frame complete. Indicates the end of the frame when the receiver is disabled. If the receive frame and clock are not disabled in the same frame, this bit is also set at the end of the frame that the receive frame and clock are disabled. See the description of SSI_CR[RCD] for more details on enabling/disabling the receive frame and clock after the receiver is disabled. 0 End of frame not reached 1 End of frame reached after clearing SSI_CR[RE], or setting SSI_CR[RCD] when RE is already cleared.
23 TFRC	Transmit frame complete. Indicates the end of the frame when the transmitter is disabled. If the transmit frame and clock are not disabled in the same frame, this bit is also set at the end of the frame that the transmit frame and clock are disabled. See the description of SSI_CR[TCD] for more details on enabling/disabling the transmit frame and clock after the transmitter is disabled. 0 End of frame not reached 1 End of frame reached after clearing SSI_CR[TE], or setting SSI_CR[RCD] when TE is already cleared.
22–19	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 29-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15
RDR1</td><td>Receive data ready 1. Only valid in two-channel mode. Indicates new data is available for the processor to read.
<table>
<tr><th rowspan="2">Rx FIFO1</th><th colspan="2">Receive data 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[RIE] set
• SSI_IER[RFF1] set</td><td>• SSI_ISR[RFF1] sets</td></tr>
<tr><td>Disabled</td><td>• SSI_IER[RIE] set
• SSI_IER[RDR1] set</td><td>• SSI_RX1 loaded with new value</td></tr>
</table>
<table>
<tr><th>Rx FIFO1</th><th>RDR1 is set when</th><th>RDR1 is cleared during any of the following</th></tr>
<tr><td>Enabled</td><td>• Rx FIFO1 loaded with new value</td><td>• Rx FIFO1 is empty
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td><td>• SSI_RX1 loaded with new value</td><td>• SSI_RX1 is read
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
<tr><td>14
RDR0</td><td>Receive data ready 0. Similar description as RDR1 but pertains to Rx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 No new data for core to read
1 New data for core to read</td></tr>
<tr><td>13
TDE1</td><td>Transmit data register empty 1. Only valid in two-channel mode. Indicates that data needs to be written to the SSI.
<table>
<tr><th rowspan="2">Tx FIFO1</th><th colspan="2">Transmit data 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[TIE] set
• SSI_IER[TDE1] set</td><td>• SSI_ISR[TDE1] sets</td></tr>
<tr><td>Disabled</td><td>• SSI_IER[TIE] set
• SSI_IER[TDE1] set</td><td>• SSI_TX1 data transferred to TXSR</td></tr>
</table>
<table>
<tr><th>Tx FIFO1</th><th>TDE1 is set when</th><th>TDE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• At least one empty slot in Tx FIFO1</td><td>• Tx FIFO1 is full
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td><td>• SSI_TX1 data transferred to TXSR</td><td>• SSI_TX1 is written
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
</table>

Table 29-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>12
TDE0</td><td>Transmit data register empty 0. Similar description as TE1 but pertains to Tx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 Data available for transmission
1 Data needs to be written by the core for transmission</td></tr>
<tr><td>11
ROE1</td><td>Receiver overrun error 1. Only valid in two-channel mode. Indicates an overrun error has occurred.
<table>
<tr><th rowspan="2">Rx FIFO1</th><th colspan="2">Receiver overrun error 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td rowspan="2">• SSI_IER[RIE] set
• SSI_IER[ROE1] set</td><td rowspan="2">• SSI_ISR[ROE1] sets</td></tr>
<tr><td>Disabled</td></tr>
</table>
<table>
<tr><th>Rx FIFO1</th><th>ROE1 is set when all of the following occur</th><th>ROE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• RXSR is full
• Rx FIFO1 is full</td><td rowspan="2">• Writing a 1 to ROE1
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td><td>• RXSR is full
• SSI_RX1 is full</td></tr>
</table>
<b>Note:</b> If Rx FIFO 1 is enabled, the RFF1 flag indicates the FIFO is full.
If Rx FIFO 1 is disabled, the RDR1 flag indicates the SSI_RX1 register is full.</td></tr>
<tr><td>10
ROE0</td><td>Receiver overrun error 0. Similar description as ROE1 but pertains to Rx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 No overrun detected
1 Receiver 0 overrun error occurred</td></tr>
<tr><td>9
TUE1</td><td>Transmitter underrun error 1. Only valid in two-channel mode. When a transmit underrun error occurs, the previous data is retransmitted. In network mode, each time slot requires data transmission (unless masked through the SSI_TMASK register), when the transmitter is enabled.
<table>
<tr><th rowspan="2">Tx FIFO1</th><th colspan="2">Transmit underrun error 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td rowspan="2">• SSI_IER[TIE] set
• SSI_IER[TUE1] set</td><td rowspan="2">• SSI_ISR[TUE1] sets</td></tr>
<tr><td>Disabled</td></tr>
</table>
<table>
<tr><th>Tx FIFO1</th><th>TUE1 is set when all of the following occur</th><th>TUE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td rowspan="2">• TXSR is empty
• SSI_ISR[TDE1] set
• Transmit time slot occurs</td><td rowspan="2">• Writing a 1 to TUE1
• SSI reset
• POR reset</td></tr>
<tr><td>Disabled</td></tr>
</table>
</td></tr>
</table>

Table 29-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>8
TUE0</td><td>Transmitter underrun error 0. Similar description as TUE1 but pertains to TDE0 and it is not necessary to be in two-channel mode for this bit to be set.
0 No underrun detected
1 Transmitter 0 underrun error occurred</td></tr>
<tr><td>7
TFS</td><td>Transmit frame sync. Indicates occurrence of a transmit frame sync during transmission of the last word written to the SSI_TX registers
<table>
<tr><th rowspan="2">SSI Mode</th><th colspan="2">Transmit frame sync interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Normal</td><td rowspan="2">• SSI_IER[TIE] set
• SSI_IER[TFS] set</td><td rowspan="2">• SSI_ISR[TFS] sets</td></tr>
<tr><td>Network</td></tr>
</table>
<table>
<tr><th>SSI Mode</th><th>TFS is set when</th><th>TFS is cleared when any of the following occur</th></tr>
<tr><td>Normal</td><td>• TFS is always set</td><td>• SSI reset
• POR reset</td></tr>
<tr><td>Network</td><td>• First time slot transmission</td><td>• Starts transmitting next time slot
• SSI reset
• POR reset</td></tr>
</table>
<b>Note:</b> Data written to the SSI_TX registers during the time slot when the TFS flag is set is sent during the second time slot (in network mode) or in the next first time slot (in normal mode).</td></tr>
<tr><td>6
RFS</td><td>Receive frame sync. Indicates occurrence of a receive frame sync during reception of the next word in SSI_RX registers.
<table>
<tr><th rowspan="2">SSI Mode</th><th colspan="2">Receive frame sync interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Normal</td><td rowspan="2">• SSI_IER[RIE] set
• SSI_IER[RFS] set</td><td rowspan="2">• SSI_ISR[RFS] sets</td></tr>
<tr><td>Network</td></tr>
</table>
<table>
<tr><th>SSI Mode</th><th>RFS is set when</th><th>RFS is cleared when any of the following occur</th></tr>
<tr><td>Normal</td><td>• RFS is always set</td><td>• SSI reset
• POR reset</td></tr>
<tr><td>Network</td><td>• First time slot received</td><td>• Starts receiving next time slot
• SSI reset
• POR reset</td></tr>
</table></td></tr>
</table>

Table 29-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
TLS
4
RLS</td><td>Transmit/receive last time slot. Indicates the current time slot is the last time slot of the frame.
<table>
<tr><th></th><th colspan="2">Last time slot interrupts</th></tr>
<tr><th></th><th>Required conditions</th><th>Trigger</th></tr>
<tr><td><b>TLS</b></td><td>• SSI_IER[TIE] set
• SSI_IER[TLS] set</td><td>• SSI_ISR[TLS] sets</td></tr>
<tr><td><b>RLS</b></td><td>• SSI_IER[RIE] set
• SSI_IER[RLS] set</td><td>• SSI_ISR[RLS] sets</td></tr>
</table>
<table>
<tr><th></th><th>Is set when</th><th>Is cleared when any of the following occur</th></tr>
<tr><td><b>TLS</b></td><td>• Start of last transmit time slot</td><td>• SSI_ISR is read with TLS set
• SSI reset
• POR reset</td></tr>
<tr><td><b>RLS</b></td><td>• End of last receive time slot</td><td>• SSI_ISR is read with RLS set
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
<tr><td>3
RFF1</td><td>Receive FIFO full 1. Only valid in two-channel mode and if Rx FIFO 1 is enabled. When Rx FIFO1 is full, all further data received (for storage in this FIFO) is ignored until the FIFO contents are read.
<table>
<tr><th>Rx FIFO1</th><th colspan="2">Receive FIFO full 1 interrupt</th></tr>
<tr><th></th><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[RIE] set
• SSI_IER[RFF1] set</td><td>• SSI_ISR[RFF1] sets</td></tr>
</table>
<table>
<tr><th>Rx FIFO1</th><th>RFF1 is set when</th><th>RFF1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• Rx FIFO 1 level reaches Rx FIFO watermark 1 (RFWM1) threshold</td><td>• Rx FIFO 1 level falls below RFWM1 level
• SSI reset
• POR reset</td></tr>
</table>
</td></tr>
<tr><td>2
RFF0</td><td>Receive FIFO full 0. Similar to description of RFF1, but pertains to Rx FIFO 0 and is not necessary to be in two-channel mode for this bit to be set.
0 Space available in receive FIFO 0
1 Receive FIFO 0 is full</td></tr>
</table>

Table 29-8. SSI_ISR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>1
TFE1</td><td>Transmit FIFO empty 1. Only valid when in two-channel mode and Tx FIFO 1 is enabled.
<table>
<tr><th rowspan="2">Tx FIFO1</th><th colspan="2">Transmit FIFO full 1 interrupt</th></tr>
<tr><th>Required conditions</th><th>Trigger</th></tr>
<tr><td>Enabled</td><td>• SSI_IER[RIE] set
• SSI_IER[TFE1] set</td><td>• SSI_ISR[TFE1] sets</td></tr>
</table>
<table>
<tr><th>Tx FIFO1</th><th>TFE1 is set when any of the following occur</th><th>TFE1 is cleared when any of the following occur</th></tr>
<tr><td>Enabled</td><td>• Tx FIFO 1 level falls below Tx FIFO watermark 1 (TFWM1) threshold
• SSI reset
• POR reset</td><td>• Tx FIFO 1 level is more than TFWM1 level</td></tr>
</table>
</td></tr>
<tr><td>0
TFE0</td><td>Transmit FIFO empty 0. Similar to description of TFE1 but pertains to TX FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.
0 Transmit FIFO 0 has data for transmission
1 Transmit FIFO 0 is empty</td></tr>
</table>

29.3.9 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	RFRC	TFRC	RDMAE	RIE	TDMAE	TIE	CMDAU	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 29-16. SSI Interrupt Enable Register (SSI_IER)

Table 29-9. SSI_IER Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24–23 RFRC TFRC	Controls if the corresponding status bit in SSI_ISR can issue an interrupt to the processor. See Section 29.3.8, “SSI Interrupt Status Register (SSI_ISR),” for details on the individual bits. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.
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 29.4.5, “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 29.4.6, “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 29.3.8, “SSI Interrupt Status Register (SSI_ISR),” for details on the individual bits. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.

29.3.10 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 29-17. SSI Transmit Configuration Register (SSI_TCR)

Table 29-10. 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 29-3 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

Table 29-10. SSI_TCR Field Descriptions (continued)

Field	Description
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

29.3.11 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 29-18. SSI Receive Configuration Register (SSI_RCR)

Table 29-11. 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

Table 29-11. SSI_RCR Field Descriptions (continued)

Field	Description
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
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.

29.3.12 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 29-19. SSI Clock Control Register (SSI_CCR)

Table 29-12. 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
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 I^{2}S master mode, the SSI works with a fixed word length of 32, and the WL bits control the amount of valid data in those 32 bits. Bits per word equal 2 × (WL + 1). Refer to the below table for details of data word lengths supported by the SSI module. **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

Table 29-12. SSI_CCR Field Descriptions (continued)

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 29.4.2.2, “DIV2, PSR and PM Bit Description,” for details regarding settings.

29.3.13 SSI FIFO Control/Status Register (SSI_FCSR)

Address: 0xFC0B_C02C (SSI_FCSR) 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	RFCNT1				TFCNT1				RFWM1				TFWM1				RFCNT0				TFCNT0				RFWM0				TFWM0			
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Figure 29-20. SSI FIFO Control/Status Register (SSI_FCSR)

Table 29-13. 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.

Table 29-13. SSI_FCSR Field Descriptions (continued)

Field	Description
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 29-14. 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

29.3.14 SSI AC97 Control Register (SSI_ACR)

SSI_ACR controls various features of the SSI operating in AC97 mode.

Address: 0xFC0B_C038 (SSI_ACR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRDIV						WR	RD	TIF	FV	AC97EN
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-21. SSI AC97 Control Register (SSI_ACR)

Table 29-15. 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.
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 29.4.1.5, “AC97 Mode,” for details of AC97 operation. 0 AC97 mode disabled 1 AC97 mode enabled

29.3.15 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

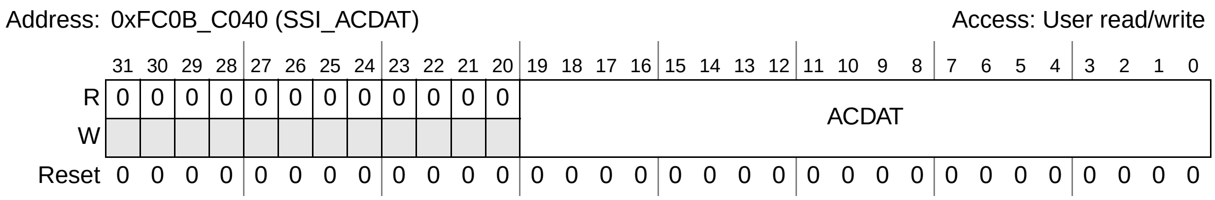
Figure 29-22. SSI AC97 Command Address Register (SSI_ACADD)

Table 29-16. 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.

29.3.16 SSI AC97 Command Data Register (SSI_ACDAT)

SSI_ACDAT contains the outgoing command data slot.



Address: 0xFC0B_C040 (SSI_ACDAT) Access: User read/write

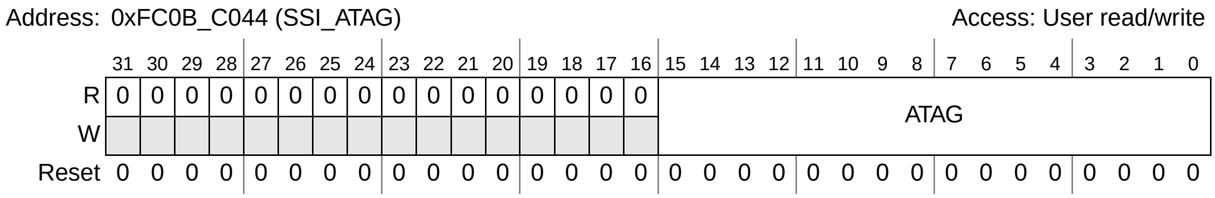
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	ACDAT																			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-23. SSI AC97 Command Data Register (SSI_ACDAT)

Table 29-17. 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.

29.3.17 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 29-24. SSI AC97 Tag Register (SSI_ATAG)

Table 29-18. 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.

29.3.18 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 W	TMASK																															
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 29-25. SSI Transmit Time Slot Mask Register (SSI_TMASK)

Table 29-19. 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)

29.3.19 SSI Receive Time Slot Mask Register (SSI_RMASK)

This register controls the time slots that the SSI receives data in network mode.

Address: 0xFC0B_C04C (SSI_RMASK) Access: User read/write

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

Figure 29-26. SSI Receive Time Slot Mask Register (SSI_RMASK)

Table 29-20. 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)

29.3.20 SSI AC97 Channel Status Register (SSI _ACCSR)

SSI _ACCSR indicates which data slot is enabled in AC97 variable mode operation.

Address: 0xFC0B_C050 (SSI_ACCSR) 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	ACCSR									
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 29-27. SSI AC97 Channel Status Register (SSI_ACCSR)

Table 29-21. SSI_ACCSR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9–0 ACCSR	AC97 channel status. Indicates which data slot is enabled in AC97 variable mode operation. This register is updated if the core enables or disables a channel through a write to SSI_ACCEN or SSI_ACCDIS or the external codec enables a channel by sending a 1 in the corresponding SLOTREQ bit. Bit 0 corresponds to the first data slot in an AC97 frame (slot #3) and bit 9 corresponds to the tenth data slot (slot #12). Writes to this register result in an error response. 0 Data channel disabled 1 Data channel enabled

29.3.21 SSI AC97 Channel Enable Register (SSI _ACCEN)

SSI _ACCEN enables data slots in AC97 variable mode operation.

Address: 0xFC0B_C054 (SSI_ACCEN) 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																							ACCEN									
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 29-28. SSI AC97 Channel Enable Register (SSI_ACCEN)

Table 29-22. SSI_ACCEN Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9–0 ACCEN	AC97 channel enable. Enables a data channel in AC97 variable mode. Writing a zero has no effect. Bit 0 corresponds to the first data slot in an AC97 frame (slot #3) and bit 9 corresponds to the tenth data slot (slot #12). These bits always read as zero. 0 No effect 1 Enables the corresponding data channel

29.3.22 SSI AC97 Channel Disable Register (SSI _ACCDIS)

SSI _ACCDIS disables data slots in AC97 variable mode operation.

Address: 0xFC0B_C058 (SSI_ACCDIS) 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																							ACCDIS									
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 29-29. SSI AC97 Channel Disable Register (SSI_ACCDIS)

Table 29-23. SSI_ACCDIS Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9–0 ACCDIS	AC97 channel disable. Disables a data channel in AC97 variable mode. Writing a zero has no effect. Bit 0 corresponds to the first data slot in an AC97 frame (slot #3) and bit 9 corresponds to the tenth data slot (slot #12). These bits always read as zero. 0 No effect 1 Disables the corresponding data channel

29.4 Functional Description

29.4.1 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.

29.4.1.1 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.

29.4.1.1.1 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.

29.4.1.1.2 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 29-30 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.

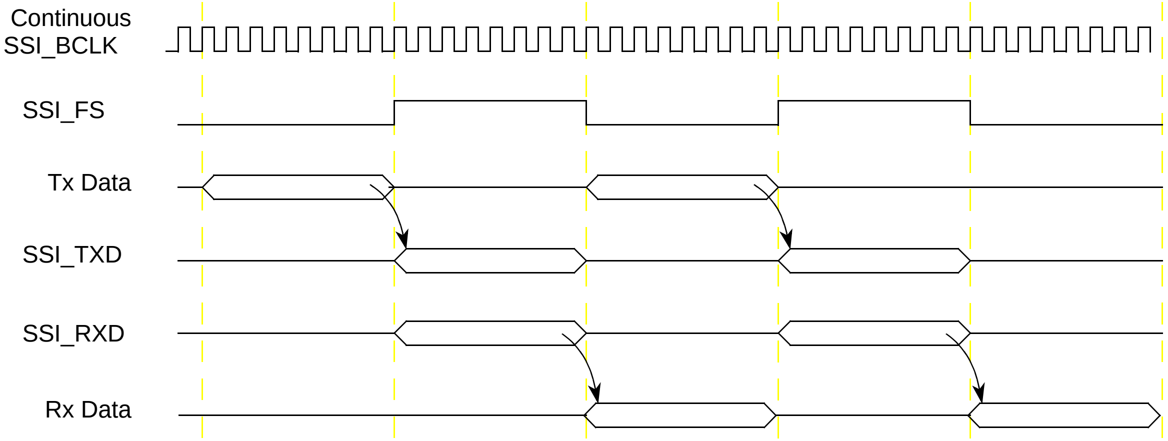


Figure 29-30. Normal Mode Timing - Continuous Clock

Figure 29-31 shows a similar case for internal (SSI generates clock) gated clock mode, and Figure 29-32 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).

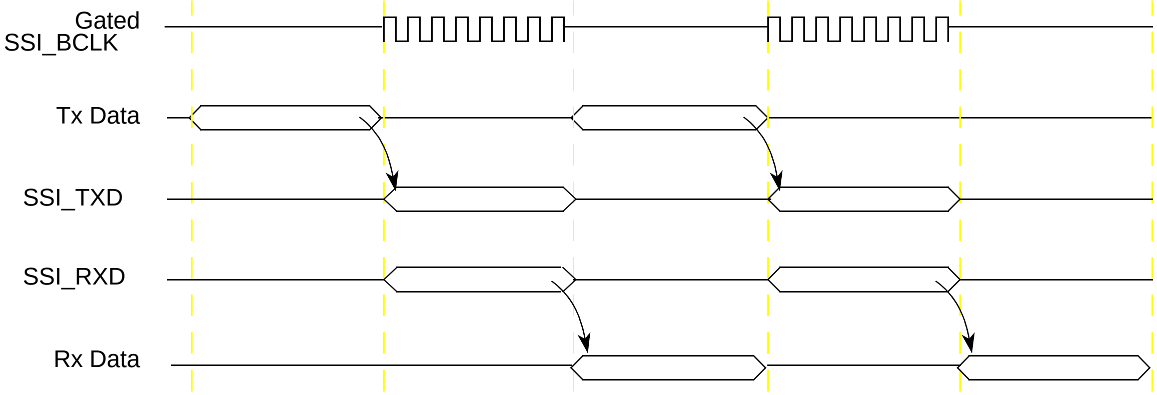


Figure 29-31. Normal Mode Timing - Internal Gated Clock

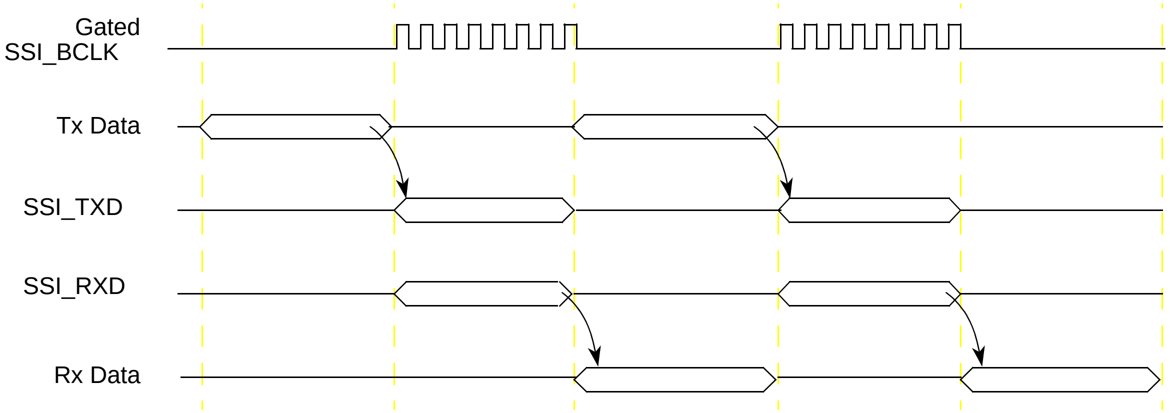


Figure 29-32. Normal Mode Timing - External Gated Clock

29.4.1.2 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 29.3.18, “SSI Transmit Time Slot Mask Register (SSI_TMASK),” and Section 29.3.19, “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).

29.4.1.2.1 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).

29.4.1.2.2 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 29-33 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 writing one to the corresponding flag in the SSI _ISR.

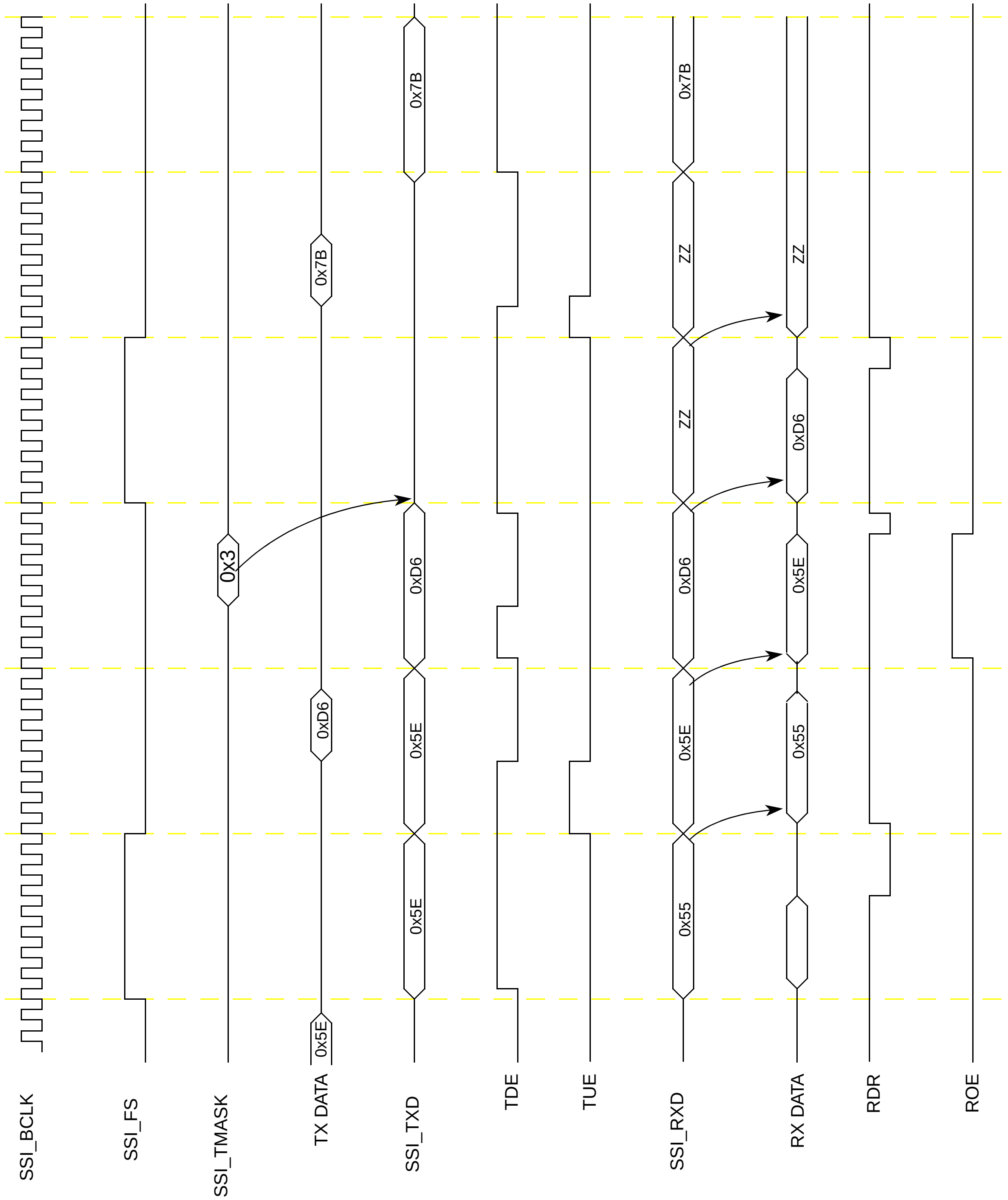


Figure 29-33. Network Mode Timing - Continuous Clock

29.4.1.3 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 29-3 for SSI configuration for gated mode operation.

The clock operates when the TE bit and/or the RE bit are appropriately enabled. For an internally generated clock, all internal bit clocks, word clocks, and frame clocks continue to operate. When a valid time slot occurs (such as the first time slot in normal mode), the internal bit clock is enabled onto the clock port. This allows data to be transferred out in periodic intervals in gated clock mode. With an external clock, the SSI module waits for a clock signal to be received. After the clock begins, valid data is shifted in. Care should be taken to clear all DC bits when the module is used in gated mode.

For gated clock operated in external clock mode, proper clock signalling must apply to SSI_BCLK for it to function properly. If the SSI uses rising edge transition to clock data (TSCKP = 0) and falling edge transition to latch data (RSCKP = 0), the clock must be in an active low state when idle. If the SSI uses falling edge transition to clock data (TSCKP = 1) and rising edge transition to latch data (RSCKP = 1), the clock must be in a active high state when idle. The following diagrams illustrate the different edge clocking/latching.

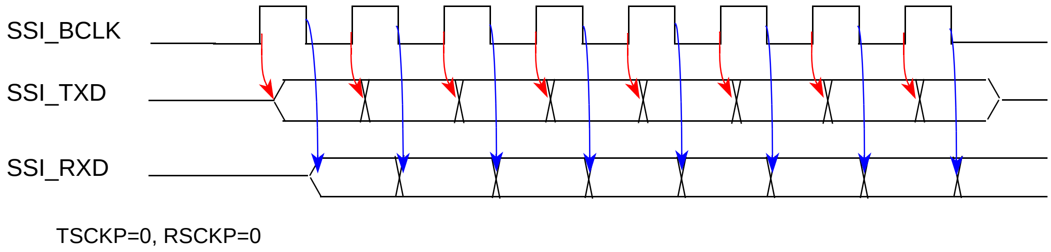


Figure 29-34. Internal Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching

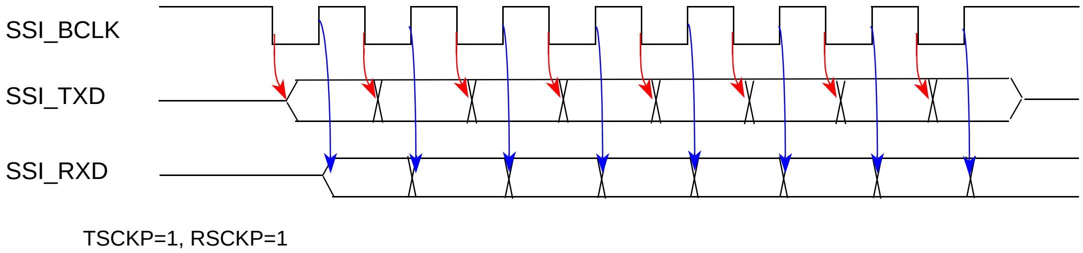


Figure 29-35. Internal Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

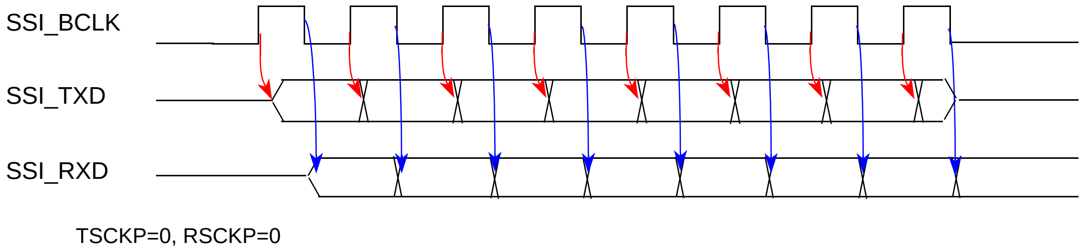


Figure 29-36. External Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching

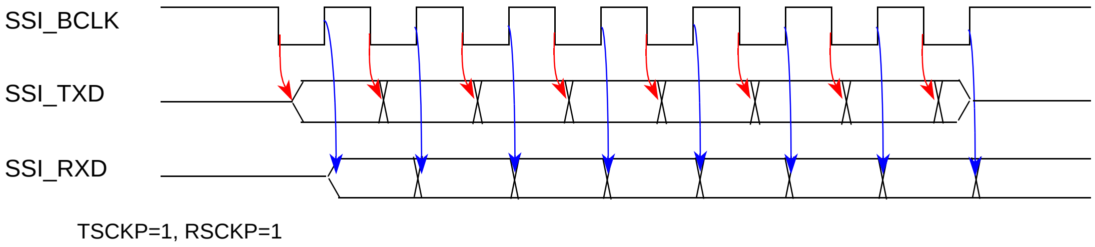


Figure 29-37. External Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

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.

29.4.1.4 I^{2}S Mode

The SSI is compliant to I^{2}S bus specification from Philips Semiconductors (February 1986, Revised June 5, 1996). Figure 29-38 depicts basic I^{2}S protocol timing.

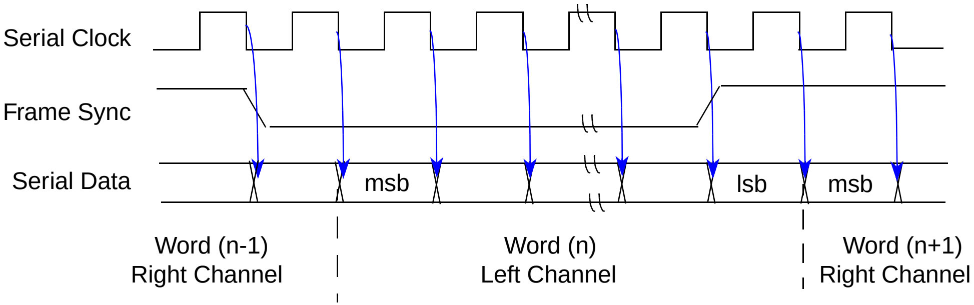


Figure 29-38. I^{2}S Mode Timing - Serial Clock, Frame Sync and Serial Data

I^{2}S mode can be selected by the SSI_CR[I2S] bits as follows:

Table 29-24. 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^{2}S modes are entered (SSI_CR[I2S] = 01 or 10), these settings are recommended:

- Synchonous mode (SSI_CR[SYN] = 1)
- Tx shift direction: msb transmitted first (SSI_TCR[TSHFD] = 0)
- Rx shift direction: msb received first (SSI_RCR[RSHFD] = 0)
- Tx data clocked at falling edge of the clock (SSI_TCR[TSCKP] = 1)
- Rx data latched at rising edge of the clock (SSI_RCR[RSCKP] = 1)
- Tx frame sync active low (SSI_TCR[TFSI] = 1)
- Rx frame sync active low (SSI_RCR[RFSI] = 1)
- Tx frame sync initiated one bit before data is transmitted (SSI_TCR[TEFS] = 1)
- Rx frame sync initiated one bit before data is received (SSI_RCR[REFS] = 1)

29.4.1.4.1 I^{2}S Master Mode

In I^{2}S master mode (SSI_CR[I2S] = 01), these additional settings are recommended:

- Internal generated bit clock (SSI_TCR[TXDIR] = 1)
- Internal generated frame sync (SSI_TCR[TFDIR] = 1)

The processor automatically performs these settings when in I^{2}S master mode:

- Network mode is selected (SSI_CR[NET] = 1)
- Tx frame sync length set to one-word-long-frame (SSI_TCR[TFSL] = 0)
- Rx frame sync length set to one-word-long-frame (SSI_RCR[RFSL] = 0)
- Tx shifting w.r.t. bit 0 of TXSR (SSI_TCR[TXBIT0] = 1)
- Rx shifting w.r.t. bit 0 of RXSR (SSI_RCR[RXBIT0] = 1)

Set the SSI_CCR[PM, PSR, DIV2, WL, DC] control bits to configure the bit clock and frame sync.

The word length is fixed to 32 in I^{2}S master mode, and the WL bits determine the number of bits that contain valid data (out of the 32 transmitted/received bits in each channel). The fixing of word duration as 32 simplifies the relation between oversampling clock (SSI_MCLK) and the frame sync (SSI_MCLK becomes an integer multiple of frame sync). The period of the oversampling clock must be at least 4x the internal bus clock period.

29.4.1.4.2 I^{2}S Slave Mode

In I^{2}S slave mode (SSI_CR[I2S] = 10), the following additional settings are recommended:

- External generated bit clock (SSI_TCR[TXDIR] = 0)
- External generated frame sync (SSI_TCR[TFDIR] = 0)

The following settings are done automatically by the processor when in I^2S 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^2S 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^2S master sends a frame sync according to the I^2S 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^2S slave mode.

29.4.1.5 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.

29.4.1.5.1 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).

29.4.1.5.2 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.

29.4.2 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^{2}S mode, the oversampling clock is available on this port if the SSI_CR[MCE] bit is set.

Figure 29-39 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 29-40.

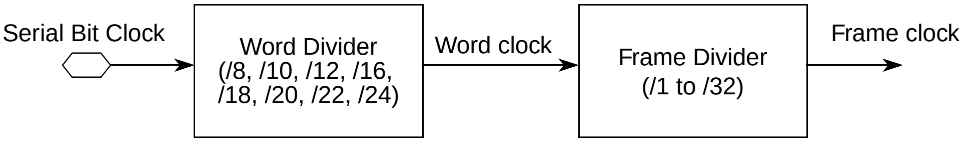


Figure 29-39. SSI Clocking

29.4.2.1 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 29-40 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.

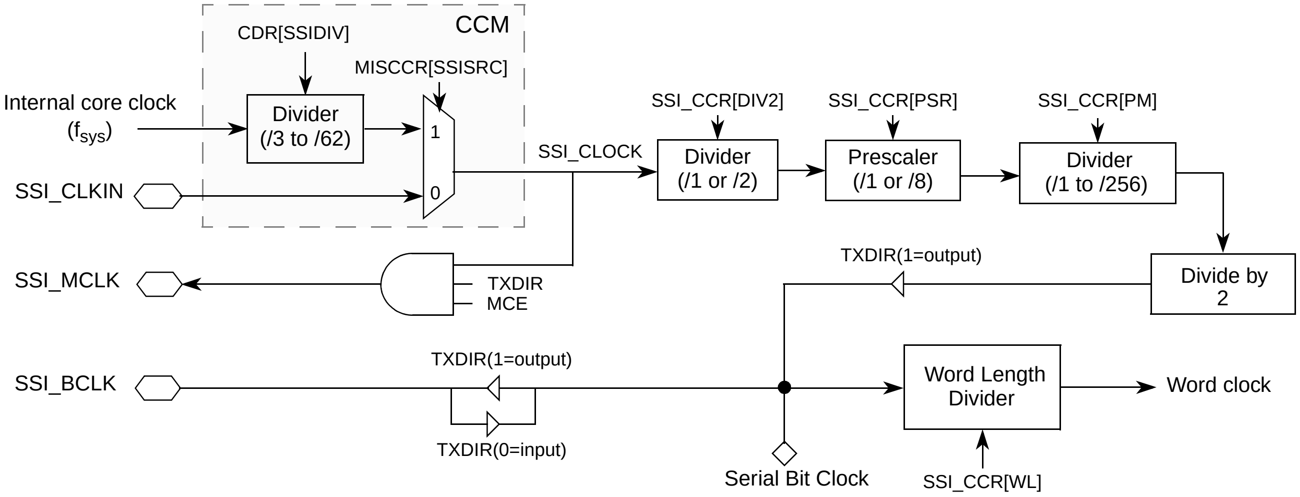


Figure 29-40. SSI Transmit Clock Generator Block Diagram

Figure 29-41 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.

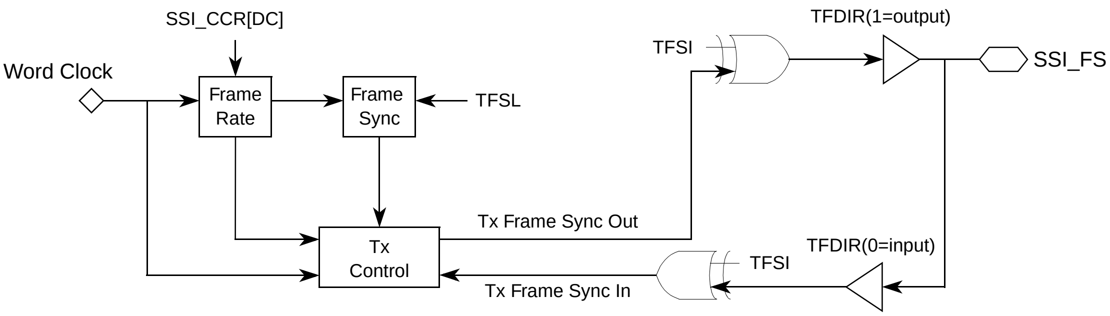


Figure 29-41. SSI Transmit Frame Sync Generator Block Diagram

29.4.2.2 DIV2, PSR and PM Bit Description

The bit clock frequency can be calculated from the SSI serial system clock (SSI_CLOCK), using Equation 29-1.

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.

$$f_{INT\_BIT\_CLK} = \frac{\text{SSI serial system clock}}{(DIV2 + 1) \times (7 \times PSR + 1) \times (PM + 1) \times 2} \qquad \textit{Eqn. 29-1}$$

From this, the frame clock frequency can be calculated:

$$f_{FS\_CLK} = \frac{f_{INT\_BIT\_CLK}}{(DC + 1) \times (2 \times (WL + 1))} \qquad \textit{Eqn. 29-2}$$

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 29-25 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 29-25. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2

SSI_CLKIN freq (MHz) (SSI_MCLK)	SSI_CCR					Bit Clk (kHz) SSI_BCLK	Frame rate (kHz)
	DIV2	PSR	PM	WL	DC		
12.288	0	0	23	3	3	256	8
12.288	0	0	11	3	3	512	16

Table 29-25. 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 29-26 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 29-26. 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

29.4.3 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.

29.4.4 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 29-27 shows the bit assignment for all the data formats supported by the SSI module.

Table 29-27. 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 29.3.10, “SSI Transmit Configuration Register (SSI_TCR),” and Section 29.3.11, “SSI Receive Configuration Register (SSI_RCR),” for more detail on the relevant bits in the SSI_TCR and SSI_RCR registers.

29.4.5 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 29-28 shows the conditions these interrupts are generated.

Table 29-28. SSI Receive Data Interrupts

Interrupt	RIE	ROE*n*	RFF*n*/RDR*n*
Receive Data 0 Interrupts (*n* = 0)			
Receive Data 0 (with exception status)	1	1	1
Receive Data 0 (without exception)	1	0	1
Receive Data 1 Interrupts (*n* = 1)			
Receive Data 1 (with exception status)	1	1	1
Receive Data 1 (without exception)	1	0	1

29.4.6 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 29-29 shows the conditions under which these interrupts are generated.

Table 29-29. 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

29.4.7 Internal Frame and Clock Shutdown

The frame sync and clock operation is determined by the SSI _CR[TCD, RCD] and SSI _CR[TE,RE] bits.

During transmit/receive operation, clearing TE/RE stops data transmission/reception when the current frame ends. If the SSI _CR[TCD or RCD] bit is set in the current or previous frames, the SSI stops driving the frame sync and clock signals when the current frame ends. After this, the SSI _ISR[TFRC, RFRC] status bits indicate the frame completion state.

Figure 29-42 illustrates a transmission case where:

- TXDIR and TFDIR are set
- TE is cleared
- TCD bit is set during the current frame

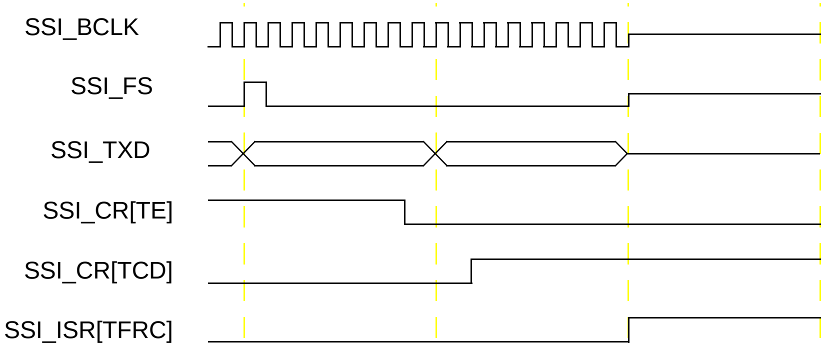


Figure 29-42. SSI _CR[TCD] Assertion in Same Frame as TE is Disabled

If SSI _CR[TCD or RCD] is not set while SSI _CR[TE or RE] is cleared, the SSI continues generating the frame sync and clock signals. Upon setting SSI _CR[TCD or RCD], the SSI stops driving these signals at the end of the current frame. Following this, the TFRC/RFRC status bits are set to indicate the frame completion state.

Figure 29-43 illustrates a transmission case where:

- TXDIR and TFDIR are set
- SSI _CR[TCD] is set a few frames after clearing TE
- SSI _ISR[TFRC] is set at the frame boundary after TE is cleared. Once software services this interrupt and later sets SSI _CR[TCD], TFRC is again set at the following frame boundary.

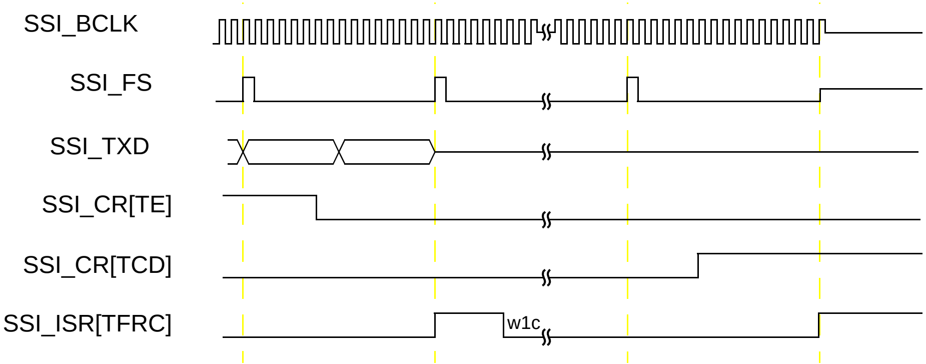


Figure 29-43. SSI _CR[TCD] Assertion in Frame After Disabling TE

29.5 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 29-4
- 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 29-30).
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 29-30.

NOTE

These control bits should not be changed when the SSI module is enabled.

Table 29-30. 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 30
Real-Time Clock

30.1 Introduction

Figure 30-1 is a block diagram of the functional organization of the real time clock (RTC) module, consisting of:

- Clock Generation
- Time-of-day (TOD) clock counter
- Alarm
- Sampling timer
- Minute stopwatch
- Associated control and bus interface hardware

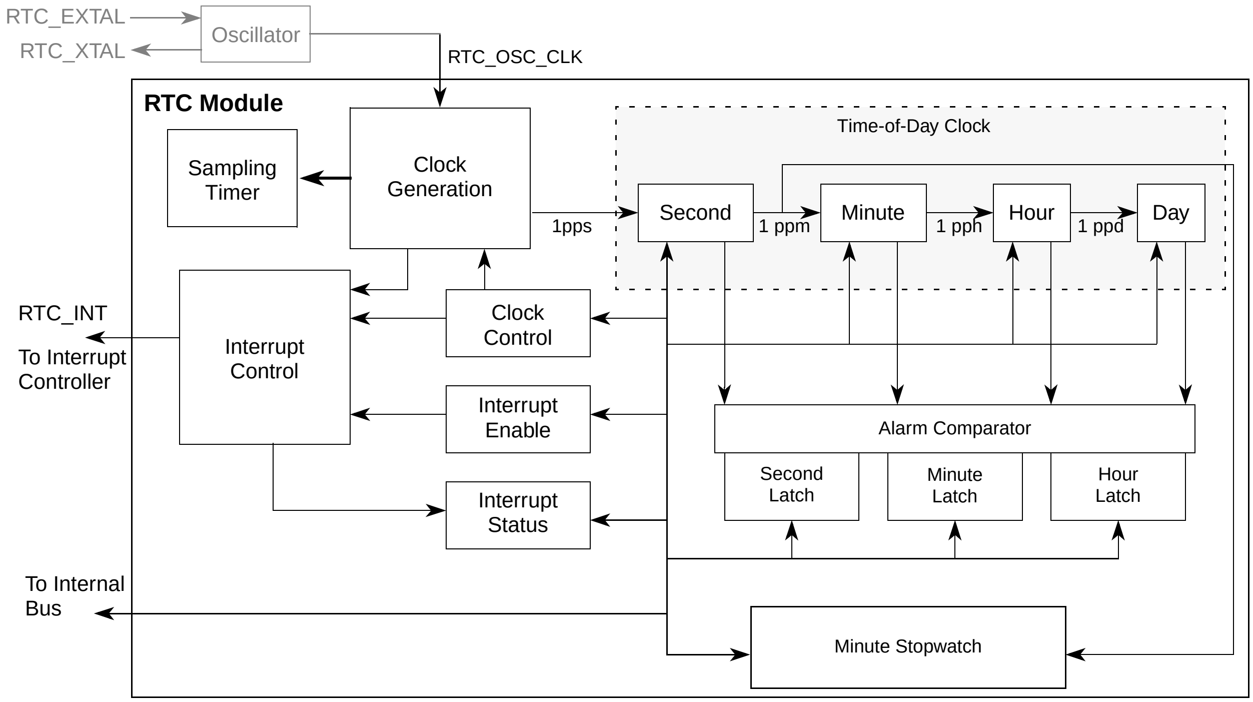


Figure 30-1. Real Time Clock Block Diagram

30.1.1 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.

30.1.2 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
- Standby operation support
- Operation determined by reference input oscillator clock frequency and value programmed into user-accessible registers
  - Minimum supported oscillator frequency is 2 Hz
- Ability to wake the processor from low-power modes (wait, doze, and stop) via the RTC interrupts

NOTE

The RTC is enabled during stop mode.

30.1.3 Modes of Operation

The real-time-clock operates in various modes as described below:

- Time-of-day counters
  - The clock generation logic divides the reference clock down to 1 Hz using the divider in the RTC_GOC register.
  - 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 clock generation logic divides the reference clock down to 512 Hz using the divider in the RTC_GOC register. 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 30-15 lists some examples of the interrupt frequencies of the sampling timer for the possible reference clocks. Sampling frequencies are dependent upon the RTC oscillator frequency and the value in RTC_GOC[31:9].
- Minute Stopwatch

— The minute stopwatch performs a countdown with a one minute resolution. It generates an interrupt on a minute boundary.

30.2 External Signal Description

The below table describes the RTC external signals.

Table 30-1. RTC Signals

Signal Name	Abbreviation	Function	I/O
RTC External Clock In	RTC_EXTAL	Crystal input clock.	I
RTC Crystal	RTC_XTAL	Oscillator output to crystal.	O
RTV Standby Voltage	VSTBY_RTC	3.3 V standby voltage supply	—

30.3 Memory Map/Register Definition

The RTC module includes eleven registers, which are summarized below.

Table 30-2. 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	30.3.1/30-3
0xFC0A_8004	RTC Seconds Counter Register (RTC_SECONDS)	32	R/W	Undefined	30.3.2/30-4
0xFC0A_8008	RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)	32	R/W	0x0000_0000	30.3.3/30-4
0xFC0A_800C	RTC Seconds Alarm Register (RTC_ALRM_SEC)	32	R/W	0x0000_0000	30.3.4/30-5
0xFC0A_8010	RTC Control Register (RTC_CR)	32	R/W	0x0000_0080	30.3.5/30-6
0xFC0A_8014	RTC Interrupt Status Register (RTC_ISR)	32	R/W	0x0000_0000	30.3.6/30-6
0xFC0A_8018	RTC Interrupt Enable Register (RTC_IER)	32	R/W	0x0000_0000	30.3.7/30-7
0xFC0A_801C	Stopwatch Minutes Register (RTC_STPWCH)	32	R/W	0x0000_003F	30.3.8/30-9
0xFC0A_8020	RTC Days Counter Register (RTC_DAYS)	32	R/W	0x0000_0000	30.3.9/30-9
0xFC0A_8024	RTC Days Alarm Register (RTC_ALRM_DAY)	32	R/W	0x0000_0000	30.3.10/30-9
0xFC0A_8034	RTC General Oscillator Clock (RTC_GOC)	32	R/W	0x0000_8000	30.3.11/30-10
0xFC0A_8038	RTC Oscillator Enable (RTC_OCEN)	32	R/W	0x0000_0000	30.3.12/30-10

30.3.1 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

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

Figure 30-2. RTC Hours and Minutes Counter Register (RTC_HOURMIN)

Table 30-3. 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).

30.3.2 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 30-3. RTC Seconds Counter Register (RTC_SECONDS)

Table 30-4. 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).

30.3.3 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 30-4. RTC Hours and Minutes Alarm Register (RTC_ALRM_HM)

Table 30-5. 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).

30.3.4 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 30-5. RTC Seconds Alarm Register (RTC_ALRM_SEC)

Table 30-6. 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).

30.3.5 RTC Control Register (RTC_CR)

The RTC_CR register enables the real-time clock module and software reset.

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	0	0	0	0	0	0	SWR
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 30-6. RTC Control Register (RTC_CR)

Table 30-7. RTC_CR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 EN	RTC enable. Enables/disables the real-time clock module. SWR has no effect on this bit. 0 Disable the RTC 1 Enable the RTC
6–1	Reserved, must be cleared.
0 SWR	Software reset. Resets the module to its default state, except for the days, hours, minutes, and seconds counters. The EN bit is also reset to its default value of one. 0 No effect 1 Reset the module

30.3.6 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 30-7. RTC Interrupt Status Register (RTC_ISR)

Table 30-8. RTC_ISR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–8 SAM*n*	Sampling timer 7–0 interrupt flags. Indicates an interrupt has occurred at the corresponding sampling rate, equal to or close to 2^{n+2} Hz depending on the combination of the RTC oscillator frequency and the programmed value in RTC_GOC[31:9]. See Section 30.4.3, “Sampling Timer,” for more details. 0 No SAM7–0 interrupt has occurred 1 A SAM7–0 interrupt has occurred
7 2HZ	2 Hz interrupt flag. Indicates an interrupt has occurred. If enabled, this bit is set every half a second. 0 No interrupt has occurred 1 A 2 Hz interrupt has occurred
6	Reserved, must be cleared.
5 HR	Hour interrupt flag. If enabled, this bit is set on every increment of the hour counter in the RTC_HOURMIN register. 0 No interrupt has occurred 1 An hour interrupt has occurred
4 1HZ	1 Hz interrupt flag. If enabled, this bit is set on every increment of the second counter of the RTC_SECONDS register. 0 No interrupt has occurred 1 A 1 Hz interrupt has occurred
3 DAY	Day interrupt flag. If enabled, this bit is set on every increment of the day counter in the RTC_DAYS register. 0 No interrupt has occurred 1 A day interrupt has occurred
2 ALM	Alarm interrupt flag. Indicates the real-time clock matches the value in the alarm registers. 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

30.3.7 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	SAM7	SAM6	SAM5	SAM4	SAM3	SAM2	SAM1	SAM0	2HZ	0	HR	1HZ	DAY	ALM	MIN	SW
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30-8. RTC Interrupt Enable Register (RTC_IER)

Table 30-9. 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.

30.3.8 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 30-9. RTC Stopwatch Minutes Register (RTC_STPWCH)

Table 30-10. 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.

30.3.9 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 30-10. RTC Days Counter Register (RTC_DAYS)

Table 30-11. 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).

30.3.10 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 30-11. RTC Day Alarm Register (RTC_ALRM_DAY)

Table 30-12. 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).

30.3.11 RTC General Oscillator Clock Register (RTC_GOC

The RTC_GOC register is the 16-bit count value used with the input RTC oscillator clock to create the 1 Hz and sample frequencies. This register can be read or written at any time. A non-zero value must be programmed into RTC_GOC for the 1 Hz internal clock to function.

Address: 0xFC0A_8034 (RTC_GOC) Access: User read/write

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

Figure 30-12. RTC General Oscillator Clock Register (RTC_GOC)

Table 30-13. RTC_GOC Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 CNT	With the oscillator clock, determines the 1 Hz and sample frequencies. A value of 1 in RTC_GOC turns off the 1 Hz signal to the counters. Writing zero to this field results in a divider value of 65,536. This field resets to 0x8000 for a 32,768 Hz oscillator clock.

30.3.12 RTC Oscillator and Clock Enable Register (RTC_OCEN)

The RTC_OCEN register enables the on-chip 32kHz oscillator and input clock.

Address: 0xFC0A_8038 (RTC_OCEN) 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	OSC BYP	CLK EN	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 30-13. RTC Oscillator Enable Register (RTC_OCEN)

Table 30-14. RTC_OCEN Field Descriptions

Field	Description
31–5	Reserved, must be cleared.
4 OSCBYP	32kHz oscillator bypass. 0 Oscillator enabled 1 Oscillator bypassed **Note:** This bit is ignored when CLKEN is cleared.
3 CLKEN	32kHz clock enable. 0 Clock disabled 1 Clock enabled
2–0	Reserved, must be cleared.

30.4 Functional Description

The clock generation logic, using the incoming RTC oscillator clock and RTC_GOC value, creates a 1 Hz signal which increments the seconds, minutes, hours, and days counters. The alarm functions, when enabled, generate RTC interrupts when the time-of-day (TOD) settings reach programmed values. The sampling timer generates fixed-frequency interrupts, and the minute stopwatch allows for efficient interrupts on minute boundaries.

30.4.1 Clock Generation and Counter

The clock generation logic divides the reference clock by the value programmed into the RTC_GOC register to obtain a 1 Hz signal. For example:

With an RTC input clock of 32 kHz, set RTC_GOC to 0x0000_7D00 (32,000).

With an RTC input clock of 48 kHz, set RTC_GOC to 0x0000_BB80 (48,000).

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.

30.4.2 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 30.5, “Initialization/Application Information,” for the correct procedure to follow when changing the alarm or time-of-day (day, hour, minute, or second) registers.

30.4.3 Sampling Timer

The sampling timer supports application software. The sampling timer generates a periodic interrupt with the frequency specified by RTC_IER[SAM*n*,2HZ]. This timer can be used for digitizer sampling, keyboard debouncing, or communication polling. The sampling timer operates only if the real-time clock is enabled and the 1 Hz signal is programmed to clock at 1 Hz. The sample clock (which is equal to SAM7) is generated by dividing the RTC oscillator frequency by the value programmed into RTC_GOC[31:9].

The following table lists example interrupt frequencies of the sampling timer for possible combinations of RTC oscillator frequency and RTC_GOC values. The following definitions apply:

RTC_OSC = RTC oscillator frequency

SAMPLE_COUNT = RTC_GOC[31:9]

Multiple RTC_IER[SAM*n*,2HZ] bits may be set and the corresponding bits in the RTC_ISR register are set at the noted frequencies.

Table 30-15. Example Sampling Timer Frequencies

Sampling Frequency	RTC_OSC = 32 kHz
	RTC_GOC = 0x7D00
	SAMPLE_COUNT = 0x3E
		RTC_OSC = 32.768 kHz
		RTC_GOC = 0x8000
		SAMPLE_COUNT = 0x40
			RTC_OSC = 38.4 kHz
			RTC_GOC = 0x9600
			SAMPLE_COUNT = 0x4B
				RTC_OSC = 48 kHz
				RTC_GOC = 0xBB80
				SAMPLE_COUNT = 0x5D

SAM7	516.13 Hz	512.00 Hz	512.00 Hz	516.13 Hz
SAM6	258.06 Hz	256.00 Hz	256.00 Hz	258.06 Hz
SAM5	129.03 Hz	128.00 Hz	128.00 Hz	129.03 Hz
SAM4	64.52 Hz	64.00 Hz	64.00 Hz	64.52 Hz
SAM3	32.26Hz	32.00 Hz	32.00 Hz	32.26Hz
SAM2	16.13 Hz	16.00 Hz	16.00 Hz	16.13
SAM1	8.06 Hz	8.00 Hz	8.00 Hz	8.06 Hz
SAM0	4.03 Hz	4.00 Hz	4.00 Hz	4.03 Hz
2HZ	2.02 Hz	2.00 Hz	2.00 Hz	2.02 Hz

30.4.4 Minute Stopwatch

The minute stopwatch performs a countdown with a one minute resolution. It can generate an interrupt on a minute boundary. For example, to turn off a peripheral after five minutes of inactivity, program a value of 0x04 into RTC_STPWCH[CNT]. At each minute, the value in the stopwatch decrements. When the stopwatch value reaches -1, interrupt occurs. The value of the register does not change until it is reprogrammed. The actual delay includes the seconds from setting the stopwatch to the next minute tick.

30.5 Initialization/Application Information

30.5.1 Flow Chart of RTC Operation

Table 30-14 shows the flow chart of a typical RTC operation.

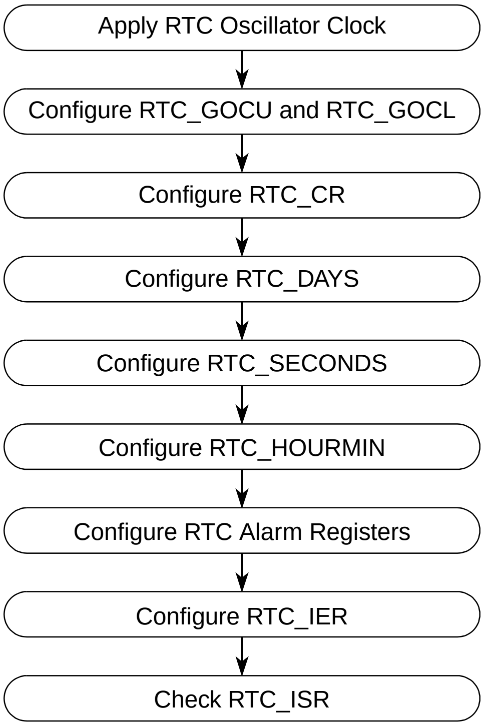


Figure 30-14. Flow Chart of RTC Operation

30.5.2 Programming the Alarm or Time-of-Day Registers

Use the following procedure illustrated in Figure 30-15 when changing the alarm or time-of-day (day, hour, minute, and second) registers.

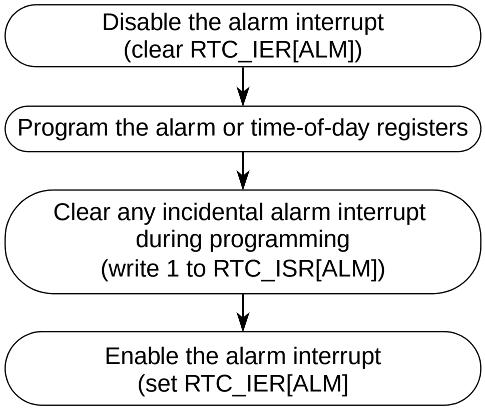

Disable the alarm interrupt
(clear RTC_IER[ALM])
Program the alarm or time-of-day registers
Clear any incidental alarm interrupt
during programming
(write 1 to RTC_ISR[ALM])
Enable the alarm interrupt
(set RTC_IER[ALM]

Figure 30-15. Flow Chart of Alarm and Time-of-Day Programming

Chapter 31
Programmable Interrupt Timers (PIT0–PIT3)

31.1 Introduction

This chapter describes the operation of the four programmable interrupt timer modules: PIT0–PIT3.

31.1.1 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.

31.1.2 Block Diagram

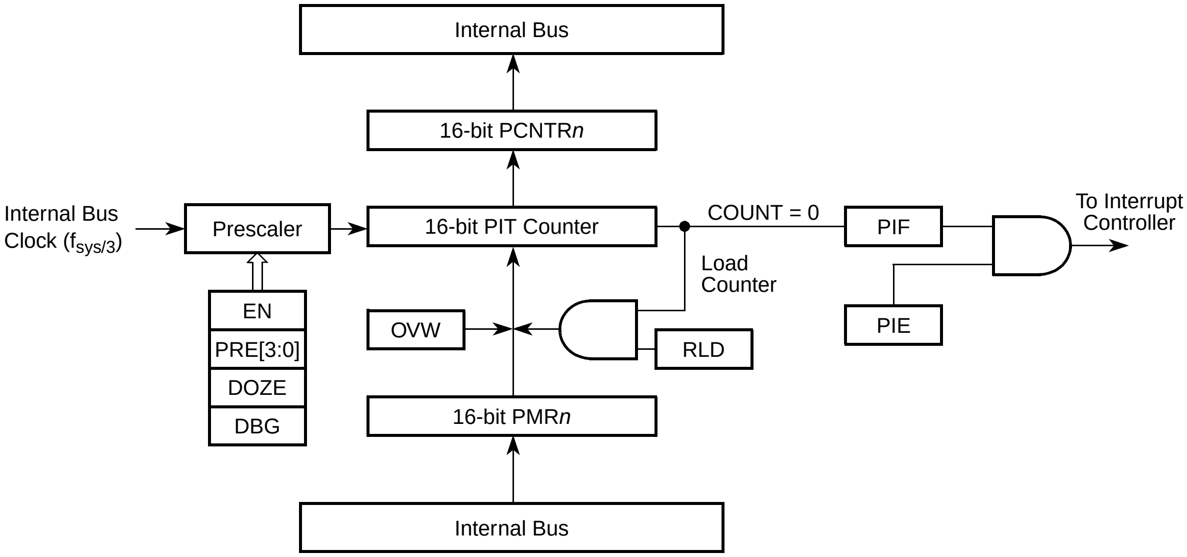


Figure 31-1. PIT Block Diagram

31.1.3 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 31-1 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 31-1. PIT Module Operation in Low-power Modes

Low-power Mode	PIT Operation	Mode Exit
Wait	Normal	N/A
Doze	Normal if PCSR*n*[DOZE] cleared, stopped otherwise	Any interrupt at or above level in LPICR, exit doze mode if PCSR*n*[DOZE] is set. Otherwise interrupt assertion has no effect.
Stop	Stopped	No
Debug	Normal if PCSR*n*[DBG] cleared, stopped otherwise	No. Any interrupt is serviced upon normal exit from debug mode

In wait mode, the PIT module continues to operate as in run mode and can be configured to exit the low-power mode by generating an interrupt request. In doze mode with the PCSR*n*[DOZE] bit set, PIT module operation stops. In doze mode with the PCSR*n*[DOZE] bit cleared, doze mode does not affect PIT operation. When doze mode is exited, PIT continues operating in the state it was in prior to doze mode. In stop mode, the internal bus clock is absent and PIT module operation stops.

In debug mode with the PCSR*n*[DBG] bit set, PIT module operation stops. In debug mode with the PCSR*n*[DBG] bit cleared, debug mode does not affect PIT operation. When debug mode is exited, the PIT continues to operate in its pre-debug mode state, but any updates made in debug mode remain.

31.2 Memory Map/Register Definition

This section contains a memory map (see Table 31-2) 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 31-2. 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 (PCSR*n*)	16	R/W	0x0000	31.2.1/31-3

Table 31-2. Programmable Interrupt Timer Modules Memory Map (continued)

Address	Register	Width (bits)	Access[1]	Reset Value	Section/Page
PIT 0 PIT 1 PIT 2 PIT 3					
0xFC08_0002 0xFC08_4002 0xFC08_8002 0xFC08_C002	PIT Modulus Register (PMR*n*)	16	R/W	0xFFFF	31.2.2/31-5
User/Supervisor Access Registers					
0xFC08_0004 0xFC08_4004 0xFC08_8004 0xFC08_C004	PIT Count Register (PCNTR*n*)	16	R	0xFFFF	31.2.3/31-5

[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.

31.2.1 PIT Control and Status Register (PCSR*n*)

The PCSR*n* registers configure the corresponding timer’s operation.

Address: 0xFC08_0000 (PCSR0)
0xFC08_4000 (PCSR1)
0xFC08_8000 (PCSR2)
0xFC08_C000 (PCSR3)

Access: Supervisor
read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	PRE				0	DOZE	DBG	OVW	PIE	PIF	RLD	EN
W														w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-2. PCSR*n* Register

Table 31-3. PCSR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–12</td><td>Reserved, must be cleared.</td></tr>
<tr><td>11–8
PRE</td><td>Prescaler. The read/write prescaler bits select the internal bus clock divisor to generate the PIT clock. To accurately predict the timing of the next count, change the PRE[3:0] bits only when the enable bit (EN) is clear. Changing PRE[3:0] resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Setting the EN bit and writing to PRE[3:0] can be done in this same write cycle. Clearing the EN bit stops the prescaler counter.
<table>
<tr><th>PRE</th><th>Internal Bus Clock Divisor</th><th>Decimal Equivalent</th></tr>
<tr><td>0000</td><td>2^{0}</td><td>1</td></tr>
<tr><td>0001</td><td>2^{1}</td><td>2</td></tr>
<tr><td>0010</td><td>2^{2}</td><td>4</td></tr>
<tr><td>...</td><td>...</td><td>...</td></tr>
<tr><td>1101</td><td>2^{13}</td><td>8192</td></tr>
<tr><td>1110</td><td>2^{14}</td><td>16384</td></tr>
<tr><td>1111</td><td>2^{15}</td><td>32768</td></tr>
</table>
</td></tr>
<tr><td>7</td><td>Reserved, must be cleared.</td></tr>
<tr><td>6
DOZE</td><td>Doze Mode Bit. The read/write DOZE bit controls the function of the PIT in doze mode. Reset clears DOZE.
0 PIT function not affected in doze mode
1 PIT function stopped in doze mode. When doze mode is exited, timer operation continues from the state it was in before entering doze mode.</td></tr>
<tr><td>5
DBG</td><td>Debug mode bit. Controls the function of PIT in halted/debug mode. Reset clears DBG. During debug mode, register read and write accesses function normally. When debug mode is exited, timer operation continues from the state it was in before entering debug mode, but any updates made in debug mode remain.
0 PIT function not affected in debug mode
1 PIT function stopped in debug mode
<b>Note:</b> Changing the DBG bit from 1 to 0 during debug mode starts the PIT timer. Likewise, changing the DBG bit from 0 to 1 during debug mode stops the PIT timer.</td></tr>
<tr><td>4
OVW</td><td>Overwrite. Enables writing to PMR<i>n</i> to immediately overwrite the value in the PIT counter.
0 Value in PMR<i>n</i> replaces value in PIT counter when count reaches 0x0000.
1 Writing PMR<i>n</i> immediately replaces value in PIT counter.</td></tr>
<tr><td>3
PIE</td><td>PIT interrupt enable. This read/write bit enables PIF flag to generate interrupt requests.
0 PIF interrupt requests disabled
1 PIF interrupt requests enabled</td></tr>
<tr><td>2
PIF</td><td>PIT interrupt flag. This read/write bit is set when PIT counter reaches 0x0000. Clear PIF by writing a 1 to it or by writing to PMR. Writing 0 has no effect. Reset clears PIF.
0 PIT count has not reached 0x0000.
1 PIT count has reached 0x0000.</td></tr>
</table>

Table 31-3. 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

31.2.2 PIT Modulus Register (PMR*n*)

The 16-bit read/write PMR*n* contains the timer modulus value loaded into the PIT counter when the count reaches 0x0000 and the PCSR*n*[RLD] bit is set.

When the PCSR*n*[OVW] bit is set, PMR*n* is transparent, and the value written to PMR*n* is immediately loaded into the PIT counter. The prescaler counter is reset (0xFFFF) anytime a new value is loaded into the PIT counter and also during reset. Reading the PMR*n* returns the value written in the modulus latch. Reset initializes PMR*n* to 0xFFFF.

Address: 0xFC08_0002 (PMR0)
0xFC08_4002 (PMR1)
0xFC08_8002 (PMR2)
0xFC08_C002 (PMR3)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PM															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 31-3. PIT Modulus Register (PMR*n)*

Table 31-4. 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.

31.2.3 PIT Count Register (PCNTR*n*)

The 16-bit, read-only PCNTR*n* contains the counter value. Reading the 16-bit counter with two 8-bit reads is not guaranteed coherent. Writing to PCNTR*n* has no effect, and write cycles are terminated normally.

Address: 0xFC08_0004 (PCNTR0)
0xFC08_4004 (PCNTR1)
0xFC08_8004 (PCNTR2)
0xFC08_C004 (PCNTR3)

Access: User read only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PC															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 31-4. PIT Count Register (PCNTR*n*)

Table 31-5. 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.

31.3 Functional Description

This section describes the PIT functional operation.

31.3.1 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.

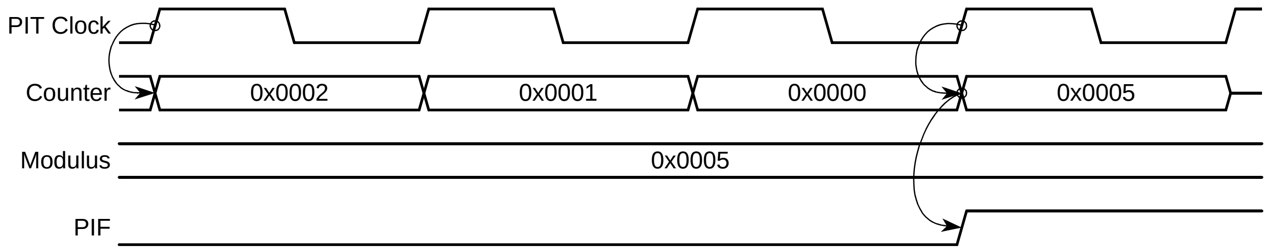


Figure 31-5. Counter Reloading from the Modulus Latch

31.3.2 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.

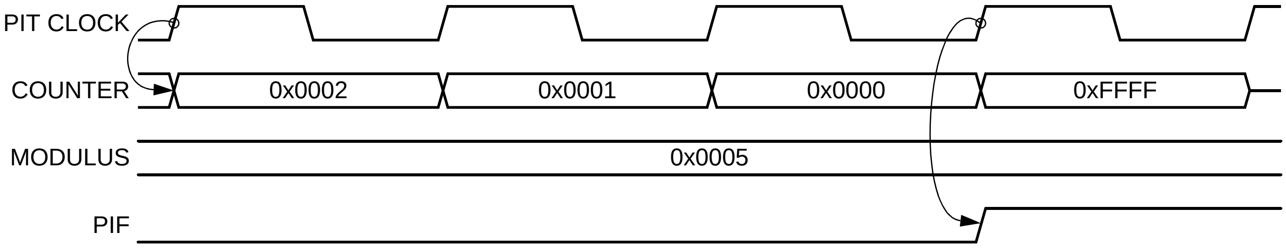


Figure 31-6. Counter in Free-Running Mode

31.3.3 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.

$$\text{Timeout period} = \frac{2^{\text{PCSRn[PRE]}} \times (\text{PMRn[PM]} + 1)}{f_{\text{sys/3}}} \qquad \textit{Eqn. 31-1}$$

31.3.4 Interrupt Operation

Table 31-6 shows the interrupt request generated by the PIT.

Table 31-6. 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 32
DMA Timers (DTIM0–DTIM3)

32.1 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.

32.1.1 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 14, “Pin-Multiplexing and Control”) prior to configuring the DMA Timers.

Figure 32-1 is a block diagram of one of the four identical timer modules.

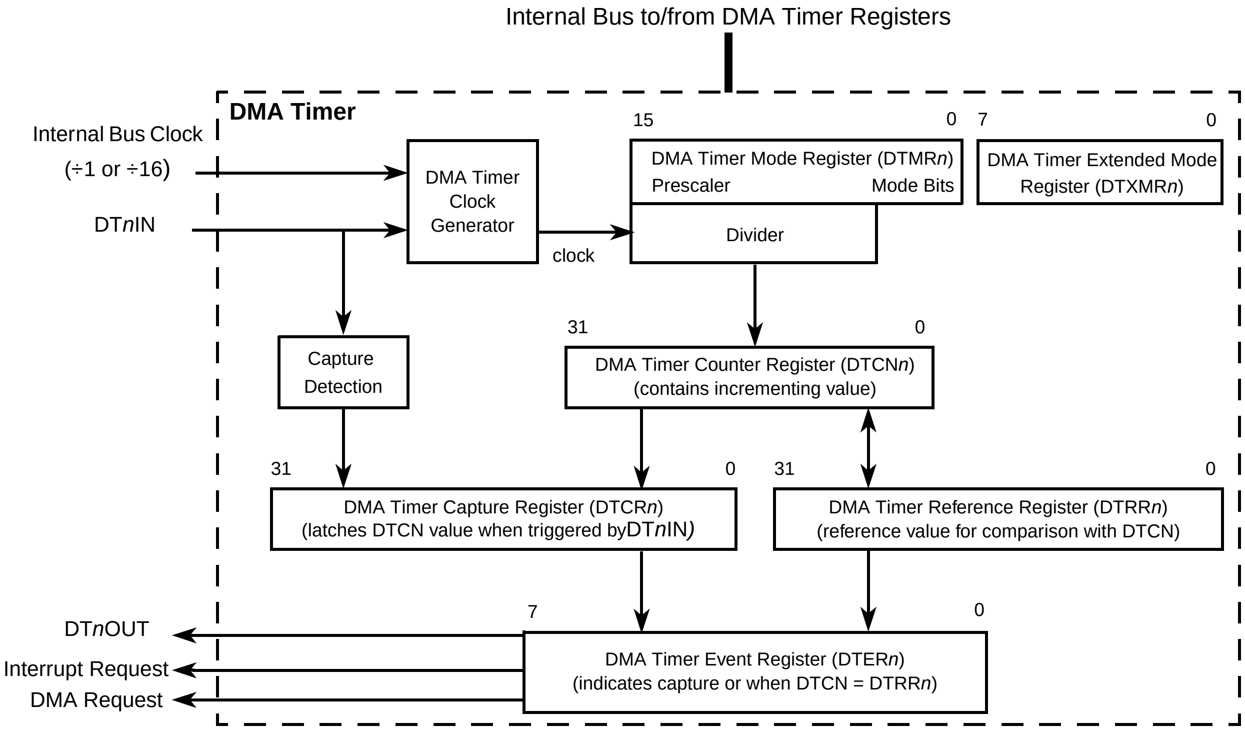

Internal Bus to/from DMA Timer Registers
DMA Timer
Internal Bus Clock
(÷1 or ÷16)
DTnIN
DMA Timer Clock Generator
clock
15
0
DMA Timer Mode Register (DTMRn)
Prescaler
Mode Bits
7
0
DMA Timer Extended Mode Register (DTXMRn)
Divider
Capture Detection
31
0
DMA Timer Counter Register (DTCNn) (contains incrementing value)
31
0
DMA Timer Capture Register (DTCRn) (latches DTCN value when triggered byDTnIN)
31
0
DMA Timer Reference Register (DTRRn) (reference value for comparison with DTCN)
7
0
DMA Timer Event Register (DTERn) (indicates capture or when DTCN = DTRRn)
DTnOUT
Interrupt Request
DMA Request

Figure 32-1. DMA Timer Block Diagram

32.1.2 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
- Ability to stop the timer from counting when the ColdFire core is halted

32.2 Memory Map/Register Definition

The timer module registers, shown in Table 32-1, can be modified at any time.

Table 32-1. DMA Timer Module Memory Map

Address	**Register**	**Width (bits)**	**Access**	**Reset Value**	**Section/Page**
DMA Timer 0 **DMA Timer 1** **DMA Timer 2** **DMA Timer 3**					
0xFC07_0000 0xFC07_4000 0xFC07_8000 0xFC07_C000	DMA Timer *n* Mode Register (DTMR*n*)	16	R/W	0x0000	32.2.1/32-3
0xFC07_0002 0xFC07_4002 0xFC07_8002 0xFC07_C002	DMA Timer *n* Extended Mode Register (DTXMR*n*)	8	R/W	0x00	32.2.2/32-5
0xFC07_0003 0xFC07_4003 0xFC07_8003 0xFC07_C003	DMA Timer *n* Event Register (DTER*n*)	8	R/W	0x00	32.2.3/32-5
0xFC07_0004 0xFC07_4004 0xFC07_8004 0xFC07_C004	DMA Timer *n* Reference Register (DTRR*n*)	32	R/W	0xFFFF_FFFF	32.2.4/32-7
0xFC07_0008 0xFC07_4008 0xFC07_8008 0xFC07_C008	DMA Timer *n* Capture Register (DTCR*n*)	32	R/W	0x0000_0000	32.2.5/32-7
0xFC07_000C 0xFC07_400C 0xFC07_800C 0xFC07_C00C	DMA Timer *n* Counter Register (DTCN*n*)	32	R	0x0000_0000	32.2.6/32-8

32.2.1 DMA Timer Mode Registers (DTMR*n*)

The DTMR*n* registers program the prescaler and various timer modes.

Address: 0xFC07_0000 (DTMR0)
0xFC07_4000 (DTMR1)
0xFC07_8000 (DTMR2)
0xFC07_C000 (DTMR3)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PS								CE		OM	ORRI	FRR	CLK		RST
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-2. DTMR*n* Registers

Table 32-2. DTMR*n* Field Descriptions

Field	Description
15–8 PS	Prescaler value. Divides the clock input (internal bus clock/(16 or 1) or clock on DT*n*IN) 0x00 1 ... 0xFF 256
7–6 CE	Capture edge. 00 Disable capture event output. Timer in reference mode. 01 Capture on rising edge only 10 Capture on falling edge only 11 Capture on any edge
5 OM	Output mode. 0 Active-low pulse for one internal bus clock cycle (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

32.2.2 DMA Timer Extended Mode Registers (DTXMR*n*)

The DTXMR*n* registers program DMA request and increment modes for the timers.

Address: 0xFC07_0002 (DTXMR0)
0xFC07_4002 (DTXMR1)
0xFC07_8002 (DTXMR2)
0xFC07_C002 (DTXMR3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	DMAEN	HALTED	0	0	0	0	0	MODE16
W								
Reset:	0	0	0	0	0	0	0	0

Figure 32-3. DTXMR*n* Registers

Table 32-3. DTXMR*n* Field Descriptions

Field	Description
7 DMAEN	DMA request. Enables DMA request output on counter reference match or capture edge event. 0 DMA request disabled 1 DMA request enabled
6 HALTED	Controls the counter when the core is halted. This allows debug mode to be entered without timer interrupts affecting the debug flow. 0 Timer function is not affected by core halt. 1 Timer stops counting while the core is halted. **Note:** This bit is only applicable in reference compare mode, see Section 32.3.3, “Reference Compare.”
5–1	Reserved, must be cleared.
0 MODE16	Selects the increment mode for the timer. Setting MODE16 is intended to exercise the upper bits of the 32-bit timer in diagnostic software without requiring the timer to count through its entire dynamic range. When set, the counter’s upper 16 bits mirror its lower 16 bits. All 32 bits of the counter remain compared to the reference value. 0 Increment timer by 1 1 Increment timer by 65,537

32.2.3 DMA Timer Event Registers (DTER*n*)

DTER*n*, shown in Figure 32-4, reports capture or reference events by setting DTER*n*[CAP] or DTER*n*[REF]. This reporting happens regardless of the corresponding DMA request or interrupt enable values, DTXMR*n*[DMAEN] and DTMR*n*[ORRI,CE].

Writing a 1 to DTER*n*[REF] or DTER*n*[CAP] clears it (writing a 0 does not affect bit value); both bits can be cleared at the same time. If configured to generate an interrupt request, clear REF and CAP early in the interrupt service routine so the timer module can negate the interrupt request signal to the interrupt controller. If configured to generate a DMA request, processing of the DMA data transfer automatically clears the REF and CAP flags via the internal DMA ACK signal.

Address: 0xFC07_0003 (DTER0)
0xFC07_4003 (DTER1)
0xFC07_8003 (DTER2)
0xFC07_C003 (DTER3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	REF	CAP
W							w1c	w1c
Reset:	0	0	0	0	0	0	0	0

Figure 32-4. DTER*n* Registers

Table 32-4. 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.
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.

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

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

32.2.4 DMA Timer Reference Registers (DTRR*n*)

As part of the output-compare function, each DTRR*n* contains the reference value compared with the respective free-running timer counter (DTCN*n*).

The reference value is matched when DTCN*n* equals DTRR*n*. The prescaler indicates that DTCN*n* should be incremented again. Therefore, the reference register is matched after DTRR*n* + 1 time intervals.

Address: 0xFC07_0004 (DTRR0)
0xFC07_4004 (DTRR1)
0xFC07_8004 (DTRR2)
0xFC07_C004 (DTRR3)

Access: User read/write

<table>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td colspan="32" rowspan="2">REF (32-bit reference value)</td></tr>
<tr><td>W</td></tr>
<tr><td>Reset</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

Figure 32-5. DTRR*n* Registers

Table 32-5. 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.

32.2.5 DMA Timer Capture Registers (DTCR*n*)

Each DTCR*n* latches the corresponding DTCN*n* value during a capture operation when an edge occurs on DT*n*IN, as programmed in DTMR*n*. The internal bus clock is assumed to be the clock source. DT*n*IN cannot simultaneously function as a clocking source and as an input capture pin. Indeterminate operation results if DT*n*IN is set as the clock source when the input capture mode is used.

Address: 0xFC07_0008 (DTCR0)
0xFC07_4008 (DTCR1)
0xFC07_8008 (DTCR2)
0xFC07_C008 (DTCR3)

Access: User read-only

<table>
<tr><td></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>R</td><td colspan="32">CAP (32-bit capture counter value)</td></tr>
<tr><td>W</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 32-6. DTCR*n* Registers

Table 32-6. 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*.

32.2.6 DMA Timer Counters (DTCN*n*)

The current value of the 32-bit timer counter can be read at anytime without affecting counting. Writes to DTCN*n* clear the timer counter. The timer counter increments on the clock source rising edge (internal bus clock divided by 1, internal bus clock divided by 16, or DT*n*IN).

Address: 0xFC07_000C (DTCN0)
0xFC07_400C (DTCN1)
0xFC07_800C (DTCN2)
0xFC07_C00C (DTCN3)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CNT (32-bit timer counter value count)																															
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 32-7. DMA Timer Counters (DTCN*n*)

Table 32-7. 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.

32.3 Functional Description

32.3.1 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*.

32.3.2 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.

32.3.3 Reference Compare

Each DMA timer can be configured to count up to a reference value. If the reference value is met, DTER*n*[REF] is set.

- If DTMR*n*[ORRI] is set and DTXMR*n*[DMAEN] is cleared, an interrupt is asserted.
- If DTMR*n*[ORRI] and DTXMR*n*[DMAEN] are set, a DMA request is asserted.

If the free run/restart bit (DTMR*n*[FRR]) is set, a new count starts. If it is clear, the timer keeps running.

32.3.4 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.

32.4 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.

32.4.1 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
```

32.4.2 Calculating Time-Out Values

Equation 32-1 determines time-out periods for various reference values:

$$\text{Timeout period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}n[\text{PS}] + 1) \times (\text{DTRR}n[\text{REF}] + 1) \qquad \textbf{\textit{Eqn. 32-1}}$$

When calculating time-out periods, add one to the prescaler to simplify calculating, because DTMR*n*[PS] equal to 0x00 yields a prescaler of one, and DTMR*n*[PS] equal to 0xFF yields a prescaler of 256.

For example, if a 80-MHz timer clock is divided by 16, DTMR*n*[PS] equals 0x7F, and the timer is referenced at 0x1312C (78,124 decimal), the time-out period is:

$$\text{Timeout period} = \frac{1}{80 \times 10^{6}} \times 16 \times (127 + 1) \times (78124 + 1) = 2.00 \text{ seconds} \qquad \textbf{\textit{Eqn. 32-2}}$$

Chapter 33
DMA Serial Peripheral Interface (DSPI)

33.1 Introduction

This chapter describes the DMA serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

33.1.1 Block Diagram

Figure 33-1 shows a block diagram of the DSPI.

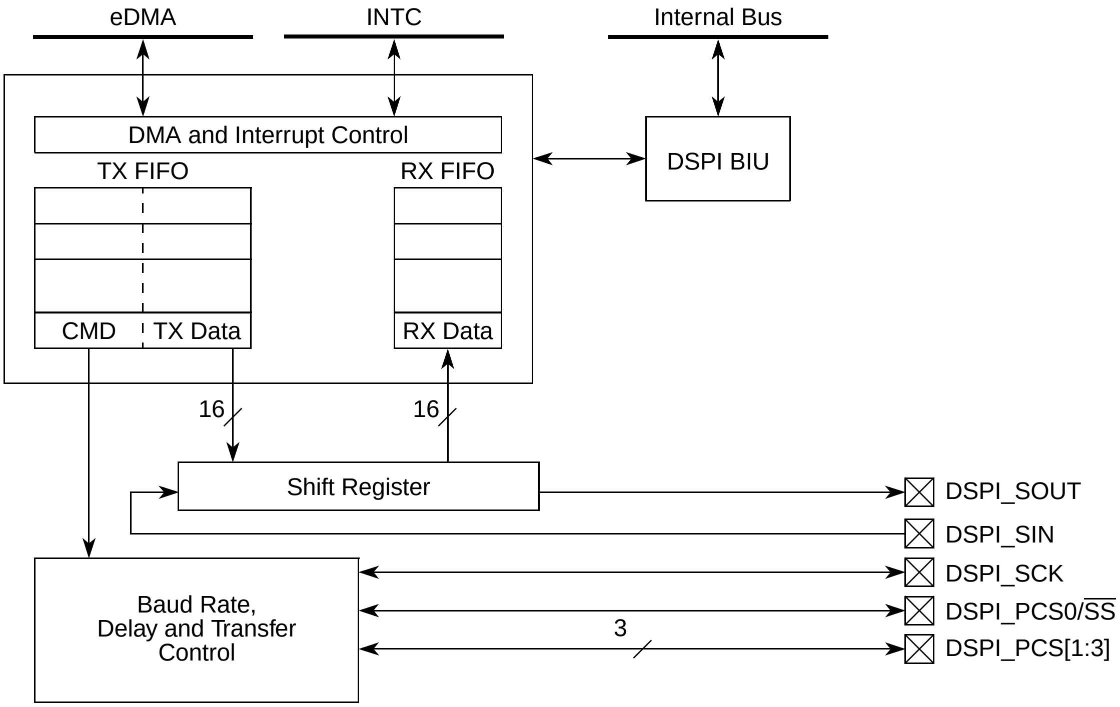


Figure 33-1. DSPI Block Diagram

33.1.2 Overview

The DMA serial peripheral interface (DSPI) block provides a synchronous serial bus for communication between an MCU and an external peripheral device. The DSPI supports up to 32 queued SPI transfers (16 receive and 16 transmit) in the DSPI resident FIFOs eliminating CPU intervention between transfers.

For queued operations, the SPI queues reside in system RAM external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of a DMA controller or through host software.

NOTE

> The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, "Pin-Multiplexing and Control") prior to configuring the DSPI.

33.1.3 Features

The DSPI module supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 16 entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - Eight clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    - PCS to SCK delay
    - SCK to PCS delay
    - Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- Four peripheral chip selects, expandable to 16 with external demultiplexer
- Two DMA conditions for SPI queues residing in RAM or Flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- Eight interrupt conditions
    - End of queue reached (EOQF)
    - TX FIFO is not full (TFFF)
    - Transfer of current frame complete (TCF)
    - FIFO underflow (slave only, the slave is asked to transfer data when the TX FIFO is empty) (TFUF)
    - RX FIFO is not empty (RFDF)
    - FIFO overflow (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF)
    - FIFO overrun (logical OR of RX overflow and TX underflow interrupts)

  - General DSPI interrupt (logical OR of the seven above conditions)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (DSPI_SCK)

33.1.4 Modes of Operation

The DSPI module has four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode.

Bits in the DSPI_MCR register determine the module-specific modes. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

33.1.4.1 Master Mode

In master mode, the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the DSPI_MCR[MSTR] bit is set. The serial communications clock (DSPI_SCK) is controlled by the master DSPI.

Master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI_CTARs sets the transfer attributes. Transfer attribute control is on a frame by frame basis. See Section 33.4.2, “Serial Peripheral Interface (SPI) Configuration” for more details.

33.1.4.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the DSPI_MCR[MSTR] bit is cleared. A bus master selects the DSPI slave by having the slave’s $\overline{\text{DSPI\_SS}}$ signal asserted. In slave mode, the bus master provides DSPI_SCK. The bus master controls all transfer attributes, but clock polarity, clock phase, and numbers of bits to transfer must be configured in the DSPI slave for proper communications.

In slave mode, data transfers MSB first. The LSBFE field of the associated CTAR register is ignored.

33.1.4.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI stops while in module disable mode. The DSPI enters the module disable mode when the DSPI_MCR[MDIS] bit is set. See Section 33.4.7, “Power Saving Features,” for more details on the module disable mode.

33.1.4.4 Debug Mode

Debug mode is used for system development and debugging. If the device enters debug mode while the DSPI_MCR[FRZ] bit is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. See Figure 33-12 for a state diagram.

33.2 External Signal Description

33.2.1 Signal Overview

Table 33-1 lists the DSPI signals.

Table 33-1. DSPI Signal Properties

Name	Function			
	Master Mode	I/O	Slave Mode	I/O
DSPI_PCS0/$\overline{\text{SS}}$	Peripheral chip select 0	Output	Slave select	Input
DSPI_PCS[1:3]	Peripheral chip select 1–3	Output	Unused	—
DSPI_SIN	Serial data in	Input	Serial data in	Input
DSPI_SOUT	Serial data out	Output	Serial data out	Output
DSPI_SCK	Serial clock	Output	Serial clock	Input

33.2.2 Peripheral Chip Select/Slave Select (DSPI_PCS0/$\overline{\text{SS}}$)

In master mode, the DSPI_PCS0 signal is a peripheral chip select output that selects the slave device to which the current transmission is intended. In slave mode, the $\overline{\text{DSPI\_SS}}$ signal is a slave select input signal allowing an SPI master to select the DSPI as the target for transmission.

33.2.3 Peripheral Chip Selects 1–3 (DSPI_PCS[1:3])

The DSPI_PCS[1:3] signals are peripheral chip select output signals in master mode. In slave mode, these signals are not used.

33.2.4 Serial Input (DSPI_SIN)

DSPI_SIN is a serial data input signal.

33.2.5 Serial Output (DSPI_SOUT)

DSPI_SOUT is a serial data output signal.

33.2.6 Serial Clock (DSPI_SCK)

DSPI_SCK is a serial communication clock signal. In master mode, DSPI generates DSPI_SCK. In slave mode, DSPI_SCK is an input from an external bus master.

33.3 Memory Map/Register Definition

Table 33-2 shows the DSPI memory map.

Table 33-2. DSPI Module Memory Map

Address	Register	Width	Access	Reset Value	Section/Page
0xFC05_C000	DSPI module configuration register (DSPI_MCR)	32	R/W	0x0000_4001	33.3.1/33-5
0xFC05_C008	DSPI transfer count register (DSPI_TCR)	32	R/W	0x0000_0000	33.3.2/33-8
0xFC05_C00C + (*n* × 0x04)	DSPI clock and transfer attributes registers (DSPI_CTAR*n*), *n*=0:7	32	R/W	0x7800_0000	33.3.3/33-8
0xFC05_C02C	DSPI status register (DSPI_SR)	32	R/W	0x0000_0000	33.3.4/33-13
0xFC05_C030	DSPI DMA/interrupt request select and enable register (DSPI_RSER)	32	R/W	0x0000_0000	33.3.5/33-15
0xFC05_C034	DSPI push TX FIFO register (DSPI_PUSHR)	32	R/W	0x0000_0000	33.3.6/33-16
0xFC05_C038	DSPI pop RX FIFO register (DSPI_POPR)	32	R	0x0000_0000	33.3.7/33-18
0xFC05_C03C + (*n* × 0x04)	DSPI transmit FIFO registers (DSPI_TXFR*n*), *n*=0:15	32	R	0x0000_0000	33.3.8/33-18
0xFC05_C07C + (*n* × 0x04)	DSPI receive FIFO registers (DSPI_RXFR*n*), *n*=0:15	32	R	0x0000_0000	33.3.9/33-19

33.3.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits that configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR may be changed while the DSPI is running.

NOTE

The DSPI_MCR[MDIS] bit is set at reset.

Address: 0xFC05_C000 (DSPI_MCR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MSTR	CONT_ SCKE	DCONF		FRZ	MTFE	0	RO OE	PCS IS7	PCS IS6	PCS IS5	PCS IS4	PCS IS3	PCS IS2	PCS IS1	PCS IS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	MDIS	DIS_ TXF	DIS_ RXF	0	0	SMPL_PT		0	0	0	0	0	0	0	HALT
W					CLR_ TXF	CLR_ RXF										
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 33-2. DSPI Module Configuration Register (DSPI_MCR)

Table 33-3. DSPI_MCR Field Descriptions

Field	Description
31 MSTR	Master/slave mode select. Configures the DSPI for master mode or slave mode. **Note:** This bit's value must only be changed when the DSPI_MCR[HALT] bit is set. Otherwise, improper operation may occur. 0 Slave mode 1 Master mode
30 CONT_ SCKE	Continuous SCK enable. Enables the serial communication clock (DSPI_SCK) to run continuously. See Section 33.4.5, "Continuous Serial Communications Clock," for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
29–28 DCONF	DSPI configuration. Selects between the different configurations of the DSPI. 00 SPI 01 Reserved 10 Reserved 11 Reserved **Note:** All values except 00 are reserved. This field must be configured for SPI mode for the DSPI module to operate correctly.
27 FRZ	Freeze. Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers 1 Halt serial transfers
26 MTFE	Modified timing format enable. Enables a modified transfer format to be used. See Section 33.4.4.4, "Modified SPI Transfer Format (MTFE = 1, CPHA = 1)," for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
25	Reserved, must be cleared.

Table 33-3. DSPI_MCR Field Descriptions (continued)

Field	Description
24 ROOE	Receive FIFO overflow overwrite enable. Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is set, incoming data is shifted into the shift register. If the ROOE bit is cleared, incoming data is ignored. See Section 33.4.6.6, "Receive FIFO Overflow Interrupt Request (RFOF)," for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
23–16 PCSIS*n*	Peripheral chip select inactive state. Determines the inactive state of the DSPI_PCS*n* signal. 0 The inactive state of DSPI_PCS*n* is low 1 The inactive state of DSPI_PCS*n* is high **Note:** DSPI_PCS7, DSPI_PCS6, and DSPI_PCS4 are not implemented on this device. Therefore, these corresponding bits are reserved. **Note:** DSPI_PCS0/$\overline{\text{SS}}$ must be configured as inactive high for slave mode operation.
15	Reserved, must be cleared.
14 MDIS	Module disable. Allows the clock to be stopped to non-memory mapped logic in DSPI effectively putting DSPI in a software controlled power-saving state. See Section 33.4.7, "Power Saving Features," for more information. This bit is set at reset. 0 Enable DSPI clocks 1 Allow external logic to disable DSPI clocks
13 DIS_TXF	Disable transmit FIFO. When the TX FIFO is disabled, transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 33.4.2.3, "FIFO Disable Operation," for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
12 DIS_RXF	Disable receive FIFO. When the RX FIFO is disabled, receive part of the DSPI operates as a simplified double-buffered SPI. See Section 33.4.2.3, "FIFO Disable Operation for details." 0 RX FIFO is enabled 1 RX FIFO is disabled
11 CLR_TXF	Clear TX FIFO. Flushes the TX FIFO. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO counter 1 Clear the TX FIFO counter **Note:** When the respective FIFO is disabled, this bit does has no effect.
10 CLR_RXF	Clear RX FIFO. Flushes the RX FIFO. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO counter 1 Clear the RX FIFO counter **Note:** When the respective FIFO is disabled, this bit does has no effect.
9–8 SMPL_PT	Sample point. Allows host software to select when the DSPI master samples SIN in modified transfer format. Figure 33-16 shows where the master can sample the SIN pin. 00 0 system clocks between DSPI_SCK edge and DSPI_SIN sample 01 1 system clock between DSPI_SCK edge and DSPI_SIN sample 10 2 system clocks between DSPI_SCK edge and DSPI_SIN sample 11 Reserved
7–1	Reserved, must be cleared.
0 HALT	Halt. Starts and stops DSPI transfers. See Section 33.4.1, "Start and Stop of DSPI Transfers," for details on the operation of this bit. 0 Start transfers 1 Stop transfers

33.3.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write to the DSPI_TCR while the DSPI is running.

Address: 0xFC05_C008 (DSPI_TCR)

Access: User read/write

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

Figure 33-3. DSPI Transfer Count Register (DSPI_TCR)

Table 33-4. DSPI_TCR Field Descriptions

Field	Description
31–16 SPI_TCNT	SPI transfer counter. Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of an SPI frame transmits. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to 0 at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter wraps around. Incrementing the counter past 65535 resets the counter to 0.
15–0	Reserved, must be cleared

33.3.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR*n*)

DSPI modules each contain eight clock and transfer attribute registers (DSPI_CTAR*n*) used to define different transfer attribute configurations. Each DSPI_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB/LSB first

DSPI_CTARs support compatibility with the QSPI module in the ColdFire family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPI_CTAR that contains the transfer’s attributes. Do not write to the DSPI_CTARs while the DSPI is running.

In master mode, the DSPI_CTAR*n* registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. When DSPI is configured as an SPI master, the DSPI_PUSHR[CTAS] field in the command portion of the TX FIFO entry selects which of the DSPI_CTAR registers is used on a per-frame basis.

In slave mode, a subset of the bit fields in the DSPI_CTAR0 registers sets the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

Address: 0xFC05_C00C (DSPI_CTAR0)
0xFC05_C010 (DSPI_CTAR1)
0xFC05_C014 (DSPI_CTAR2)
0xFC05_C018 (DSPI_CTAR3)
0xFC05_C01C (DSPI_CTAR4)
0xFC05_C020 (DSPI_CTAR5)
0xFC05_C024 (DSPI_CTAR6)
0xFC05_C028 (DSPI_CTAR7)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	DBR	FMSZ				CPOL	CPHA	LSB FE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-4. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR*n*)

Table 33-5. DSPI_CTAR*n* Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31
DBR</td><td>Double baud rate. The DBR bit doubles the effective baud rate of the serial communications clock (SCK). This field is only used in master mode. It effectively halves the baud rate division ratio supporting faster frequencies and odd division ratios for the serial communications clock (SCK). When the DBR bit is set, the duty cycle of the serial communications clock (SCK) depends on the value in the baud rate prescaler and the clock phase bit as listed below. See the BR field below and Section 33.4.3.1, “Baud Rate Generator” for details on how to compute the baud rate. If the overall baud rate is divided by two or three of the system clock, the continuous SCK enable or the modified timing format enable bits must not be set.
0 The baud rate is computed normally with a 50/50 duty cycle
1 Baud rate is doubled with the duty cycle depending on the baud rate prescaler
<table>
<tr><th>DBR</th><th>CPHA</th><th>PBR</th><th>SCK Duty Cycle</th></tr>
<tr><td>0</td><td>any</td><td>any</td><td>50/50</td></tr>
<tr><td>1</td><td>0</td><td>00</td><td>50/50</td></tr>
<tr><td>1</td><td>0</td><td>01</td><td>33/66</td></tr>
<tr><td>1</td><td>0</td><td>10</td><td>40/60</td></tr>
<tr><td>1</td><td>0</td><td>11</td><td>43/57</td></tr>
<tr><td>1</td><td>1</td><td>00</td><td>50/50</td></tr>
<tr><td>1</td><td>1</td><td>01</td><td>66/33</td></tr>
<tr><td>1</td><td>1</td><td>10</td><td>60/40</td></tr>
<tr><td>1</td><td>1</td><td>11</td><td>57/43</td></tr>
</table>
</td></tr>
</table>

Table 33-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>30–27
FMSZ</td><td>Frame size. Selects the number of bits transferred per frame. The FMSZ field is used in master mode and slave mode. The table below lists the frame sizes.
<table>
<tr><th>FMSZ</th><th>Framesize</th><th>FMSZ</th><th>Framesize</th></tr>
<tr><td>0000</td><td>Reserved</td><td>1000</td><td>9</td></tr>
<tr><td>0001</td><td>Reserved</td><td>1001</td><td>10</td></tr>
<tr><td>0010</td><td>Reserved</td><td>1010</td><td>11</td></tr>
<tr><td>0011</td><td>4</td><td>1011</td><td>12</td></tr>
<tr><td>0100</td><td>5</td><td>1100</td><td>13</td></tr>
<tr><td>0101</td><td>6</td><td>1101</td><td>14</td></tr>
<tr><td>0110</td><td>7</td><td>1110</td><td>15</td></tr>
<tr><td>0111</td><td>8</td><td>1111</td><td>16</td></tr>
</table></td></tr>
<tr><td>26
CPOL</td><td>Clock polarity. Selects the inactive state of the serial communications clock (DSPI_SCK). This bit is used in master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the continuous selection format is selected (CONT or DCONT is set), switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. For more information on continuous selection format, refer to Section 33.4.4.5, “Continuous Selection Format.”
0 The inactive state value of DSPI_SCK is low
1 The inactive state value of DSPI_SCK is high</td></tr>
<tr><td>25
CPHA</td><td>Clock phase. Selects which edge of DSPI_SCK causes data to change and which edge causes data to be captured. This bit is used in master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings.
<b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching between clock phases without stopping the DSPI can cause errors in the transfer.
0 Data is captured on the leading edge of DSPI_SCK and changed on the following edge
1 Data is changed on the leading edge of DSPI_SCK and captured on the following edge</td></tr>
<tr><td>24
LSBFE</td><td>LSB first enable. Selects if the LSB or MSB of the frame is transferred first. This bit is only used in master mode.
0 Data is transferred MSB first
1 Data is transferred LSB first</td></tr>
<tr><td>23–22
PCSSCK</td><td>PCS to SCK delay prescaler. Selects the prescaler value for the delay between assertion of DSPI_PCS and the first edge of the DSPI_SCK. This field is only used in master mode.
<b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching the PCS to SCK delay prescaler without stopping the DSPI can cause errors in the transfer.
<b>Note:</b> See Section 33.4.3.2, “PCS to SCK Delay (tCSC),” for details on calculating the PCS to SCK delay.
00 1 clock DSPI_PCS to DSPI_SCK delay prescaler
01 3 clock DSPI_PCS to DSPI_SCK delay prescaler
10 5 clock DSPI_PCS to DSPI_SCK delay prescaler
11 7 clock DSPI_PCS to DSPI_SCK delay prescaler</td></tr>
</table>

Table 33-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>21–20
PASC</td><td>After SCK delay prescaler. Selects the prescaler value for the delay between the last edge of DSPI_SCK and the negation of DSPI_PCS. This field is only used in master mode. The ASC field description in Table 33-5 explains how to compute the after SCK delay.
00 1 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler
01 3 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler
10 5 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler
11 7 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler</td></tr>
<tr><td>19–18
PDT</td><td>Delay after transfer prescaler. The PDT field selects the prescaler value for the delay between the negation of the DSPI_PCS signal at the end of a frame and the assertion of DSPI_PCS at the beginning of the next frame. The PDT field is only used in master mode. The DT field description in Table 33-5 explains how to compute the delay after transfer.
00 1 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler
01 3 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler
10 5 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler
11 7 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler</td></tr>
<tr><td>17–16
PBR</td><td>Baud rate prescaler. Selects the prescaler value for the baud rate. This field is only used in master mode. The baud rate is the frequency of the serial communications clock (DSPI_SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The description for Section 33.4.3.1, “Baud Rate Generator” details how to compute the baud rate.
00 2 clock prescaler to divide system clock
01 3 clock prescaler to divide system clock
10 5 clock prescaler to divide system clock
11 7 clock prescaler to divide system clock</td></tr>
<tr><td>15–12
CSSCK</td><td>PCS to SCK delay scaler. Selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK delay is the delay between the assertion of DSPI_PCS and the first edge of the DSPI_SCK. The table below lists the scaler values.
<b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching the PCS to SCK delay prescaler without stopping the DSPI can cause errors in the transfer.
<table>
<tr><th>CSSCK</th><th>PCS to SCK Delay Scaler Value</th><th>CSSCK</th><th>PCS to SCK Delay Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
<b>Note:</b> See Section 33.4.3.2, “PCS to SCK Delay (tCSC),” for details on calculating the PCS to SCK delay.</td></tr>
</table>

Table 33-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>11–8
ASC</td><td>After SCK delay scaler. Selects the scaler value for the after SCK delay. This field is only used in master mode. The after SCK delay is the delay between the last edge of DSPI_SCK and the negation of DSPI_PCS. The table below lists the scaler values.
<table>
<tr><th>ASC</th><th>After SCK Delay Scaler Value</th><th>ASC</th><th>After SCK Delay Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
<b>Note:</b> See Section 33.4.3.3, “After SCK Delay (tASC),” for more details on calculating the after SCK delay.</td></tr>
<tr><td>7–4
DT</td><td>Delay after transfer scaler. The DT field selects the delay after transfer scaler. This field is only used in master mode. The delay after transfer is the time between the negation of the DSPI_PCS signal at the end of a frame and the assertion of DSPI_PCS at the beginning of the next frame. The table below lists the scaler values.
<table>
<tr><th>DT</th><th>Delay after Transfer Scaler Value</th><th>DT</th><th>Delay after Transfer Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>512</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>1024</td></tr>
<tr><td>0010</td><td>8</td><td>1010</td><td>2048</td></tr>
<tr><td>0011</td><td>16</td><td>1011</td><td>4096</td></tr>
<tr><td>0100</td><td>32</td><td>1100</td><td>8192</td></tr>
<tr><td>0101</td><td>64</td><td>1101</td><td>16384</td></tr>
<tr><td>0110</td><td>128</td><td>1110</td><td>32768</td></tr>
<tr><td>0111</td><td>256</td><td>1111</td><td>65536</td></tr>
</table>
<b>Note:</b> See Section 33.4.3.4, “Delay after Transfer (tDT),” for more details on calculating the delay after transfer.</td></tr>
</table>

Table 33-5. DSPI_CTAR*n* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–0
BR</td><td>Baud rate scaler. Selects the scaler value for the baud rate. This field is only used in master mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the DSPI_SCK. The table below lists the baud rate scaler values.
<table>
<tr><th>BR</th><th>Baud Rate Scaler Value</th><th>BR</th><th>Baud Rate Scaler Value</th></tr>
<tr><td>0000</td><td>2</td><td>1000</td><td>256</td></tr>
<tr><td>0001</td><td>4</td><td>1001</td><td>512</td></tr>
<tr><td>0010</td><td>6</td><td>1010</td><td>1024</td></tr>
<tr><td>0011</td><td>8</td><td>1011</td><td>2048</td></tr>
<tr><td>0100</td><td>16</td><td>1100</td><td>4096</td></tr>
<tr><td>0101</td><td>32</td><td>1101</td><td>8192</td></tr>
<tr><td>0110</td><td>64</td><td>1110</td><td>16384</td></tr>
<tr><td>0111</td><td>128</td><td>1111</td><td>32768</td></tr>
</table>
<b>Note:</b> See Section 33.4.3.1, “Baud Rate Generator,” for more details on calculating the baud rate.</td></tr>
</table>

33.3.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a 1 to it. Writing a 0 to a flag bit has no effect.

Address: 0xFC05_C02C (DSPI_SR) Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-5. DSPI Status Register (DSPI_SR)

Table 33-6. DSPI_SR Field Descriptions

Field	Description
31 TCF	Transfer complete flag. Indicates all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the t_{ASC} delay starts. Refer to Section 33.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete
30 TXRXS	TX and RX status. Reflects the status of the DSPI. See Section 33.4.1, “Start and Stop of DSPI Transfers” for information on what causes this bit to be cleared or set. 0 TX and RX operations are disabled (DSPI is in stopped state) 1 TX and RX operations are enabled (DSPI is in running state)
29	Reserved, must be cleared.
28 EOQF	End of queue flag. Indicates transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the t_{ASC} delay starts. Refer to Section 33.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” for details. The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing SPI command 1 EOQ bit is set in the executing SPI command **Note:** EOQF does not function in slave mode.
27 TFUF	Transmit FIFO underflow flag. Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
26	Reserved, must be cleared.
25 TFFF	Transmit FIFO fill flag. Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. Therefore, this bit is set after DSPI_MCR[MDIS] is cleared after a reset. The TFFF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full
24–20	Reserved, must be cleared.
19 RFOF	Receive FIFO overflow flag. Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
18	Reserved, must be cleared.
17 RFDF	Receive FIFO drain flag. Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty **Note:** In the interrupt service routine, RFDF must be cleared only after the DSPI_POPR register is read.
16	Reserved, must be cleared.

Table 33-6. DSPI_SR Field Descriptions (continued)

Field	Description
15–12 TXCTR	TX FIFO counter. Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
11–8 TXNXTPTR	Transmit next pointer. Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 33.4.2.4, “TX FIFO Buffering Mechanism,” for more details.
7–4 RXCTR	RX FIFO counter. Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI_POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the t_{ASC} delay starts. Refer to Section 33.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” for details.
3–0 POPNXTPTR	Pop next pointer. Contains a pointer to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 33.4.2.5, “RX FIFO Buffering Mechanism” for more details.

33.3.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER serves two purposes. It enables flag bits in the DSPI_SR to generate DMA requests or interrupt requests. The DSPI_RSER also selects the type of request to be generated. See the individual bit descriptions for information on the types of requests the bits support. Do not write to the DSPI_RSER while the DSPI is running.

Address: 0xFC05_C030 (DSPI_RSER) Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-6. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 33-7. DSPI_RSER Field Descriptions

Field	Description
31 TCF_RE	Transmission complete request enable. Enables DSPI_SR[TCF] flag to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
30–29	Reserved, must be cleared.

Table 33-7. DSPI_RSER Field Descriptions (continued)

Field	Description
28 EOQF_RE	DSPI finished request enable. Enables the DSPI_SR[EOQF] flag to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
27 TFUF_RE	Transmit FIFO underflow request enable. Enables the DSPI_SR[TFUF] flag to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
26	Reserved, must be cleared.
25 TFFF_RE	Transmit FIFO fill request enable. Enables the DSPI_SR[TFFF] flag to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt or DMA requests are disabled 1 TFFF interrupt or DMA requests are enabled
24 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the DSPI_SR[TFFF] flag bit and the DSPI_RSER[TFFF_RE] bit are set, this bit selects between generating an interrupt request or a DMA request. 0 TFFF flag generates interrupt requests 1 TFFF flag generates DMA requests
23–20	Reserved, must be cleared.
19 RFOF_RE	Receive FIFO overflow request enable. Enables the DSPI_SR[RFOF] flag to generate an interrupt request. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
18	Reserved, must be cleared.
17 RFDF_RE	Receive FIFO drain request enable. Enables the DSPI_SR[RFDF] flag to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt or DMA requests are disabled 1 RFDF interrupt or DMA requests are enabled
16 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the DSPI_SR[RFDF] flag bit and the DSPI_RSER[RFDF_RE] bit are set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF flag generates interrupt requests 1 RFDF flag generates DMA requests
15–0	Reserved, must be cleared.

33.3.6 DSPI Push Transmit FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides a means to write to the TX FIFO. SPI commands and data written to this register is transferred to the TX FIFO. See Section 33.4.2.4, “TX FIFO Buffering Mechanism,” for more information. Write accesses of 8- or 16-bits to the DSPI_PUSHR transfer 32 bits to the TX FIFO.

NOTE

Only the TXDATA field is used for DSPI slaves.

Address: 0xFC05_C034 (DSPI_PUSHR) Access: User Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CONT	CTAS			EOQ	CT CNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-7. DSPI Push Transmit FIFO Register (DSPI_PUSHR)

Table 33-8. DSPI_PUSHR Field Descriptions

Field	Description
31 CONT	Continuous peripheral chip select enable. Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 33.4.4.5, “Continuous Selection Format,” for more information. 0 Return DSPI_PCS*n* signals to their inactive state between transfers 1 Keep DSPI_PCS*n* signals asserted between transfers
30–28 CTAS	Clock and transfer attributes select. Selects which of the DSPI_CTAR*n* registers is used to set the transfer attributes for the associated SPI frame. This field is used only in SPI master mode. In SPI slave mode, DSPI_CTAR0 is used instead. 000 DSPI_CTAR0 001 DSPI_CTAR1 010 DSPI_CTAR2 011 DSPI_CTAR3 100 DSPI_CTAR4 101 DSPI_CTAR5 110 DSPI_CTAR6 111 DSPI_CTAR7
27 EOQ	End of queue. Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the DSPI_SR[EOQF] bit is set. This bit is used only in SPI master mode. 0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer
26 CTCNT	Clear SPI_TCNT. Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the DSPI_TCR[SPI_TCNT] field. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. This bit is used only in SPI master mode. 0 Do not clear DSPI_TCR[SPI_TCNT] field 1 Clear DSPI_TCR[SPI_TCNT] field
25–24	Reserved, must be cleared.

Table 33-8. DSPI_PUSHR Field Descriptions (continued)

Field	Description
23–16 PCS*n*	Peripheral chip select *n*. Selects which DSPI_PCS*n* signals are asserted for the transfer. This bit is used only in SPI master mode. 0 Negate the DSPI_PCS*n* signal 1 Assert the DSPI_PCS*n* signal **Note:** DSPI_PCS7, DSPI_PCS6, and DSPI_PCS4 are not implemented on this device. Therefore, these corresponding bits are reserved.
15–0 TXDATA	Transmit data. Holds SPI data to be transferred according to the associated SPI command. **Note:** TXDATA is used in slave mode.

33.3.7 DSPI Pop Receive FIFO Register (DSPI_POPR)

The DSPI_POPR provides a means to read the RX FIFO. See Section 33.4.2.5, “RX FIFO Buffering Mechanism” for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI_POPR read from the RX FIFO and update the counter and pointer.

Address: 0xFC05_C038 (DSPI_POPR) Access: User read-only

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

Figure 33-8. DSPI Pop Receive FIFO Register (DSPI_POPR)

Table 33-9. DSPI_POPR Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 RXDATA	Received data. Contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (DSPI_SR[POPNXTPTR]).

33.3.8 DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR*n*)

The DSPI_TXFR*n* registers provide visibility into TX FIFO for debugging purposes. Each register is an entry in TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFR*n* registers does not alter the state of TX FIFO. The 16-entry deep FIFO is implemented with 16 registers, DSPI_TXFR0–15.

Address: 0xFC05_C03C (DSPI_TXFR0) 0xFC05_C05C (DSPI_TXFR8)
0xFC05_C040 (DSPI_TXFR1) 0xFC05_C060 (DSPI_TXFR9)
0xFC05_C044 (DSPI_TXFR2) 0xFC05_C064 (DSPI_TXFR10)
0xFC05_C048 (DSPI_TXFR3) 0xFC05_C068 (DSPI_TXFR11)
0xFC05_C04C (DSPI_TXFR4) 0xFC05_C06C (DSPI_TXFR12)
0xFC05_C050 (DSPI_TXFR5) 0xFC05_C070 (DSPI_TXFR13)
0xFC05_C054 (DSPI_TXFR6) 0xFC05_C074 (DSPI_TXFR14)
0xFC05_C058 (DSPI_TXFR7) 0xFC05_C078 (DSPI_TXFR15)

Access: User read-only

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

Figure 33-9. DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR*n*)

Table 33-10. DSPI_TXFR*n* Field Descriptions

Field	Description
31–16 TXCMD	Transmit command. Contains the command that sets the transfer attributes for the SPI data. See Section 33.3.6, “DSPI Push Transmit FIFO Register (DSPI_PUSHR),” for details on the command field.
15–0 TXDATA	Transmit data. Contains the SPI data to be shifted out.

33.3.9 DSPI Receive FIFO Registers 0–15 (DSPI_RXFR*n*)

The DSPI_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFR*n* registers does not alter the state of the RX FIFO. The device uses 16 registers to implement the RX FIFO; DSPI_RXFR0–15 are used.

Address: 0xFC05_C07C (DSPI_RXFR0) 0xFC05_C09C (DSPI_RXFR8)
0xFC05_C080 (DSPI_RXFR1) 0xFC05_C0A0 (DSPI_RXFR9)
0xFC05_C084 (DSPI_RXFR2) 0xFC05_C0A4 (DSPI_RXFR10)
0xFC05_C088 (DSPI_RXFR3) 0xFC05_C0A8 (DSPI_RXFR11)
0xFC05_C08C (DSPI_RXFR4) 0xFC05_C0AC (DSPI_RXFR12)
0xFC05_C090 (DSPI_RXFR5) 0xFC05_C0B0 (DSPI_RXFR13)
0xFC05_C094 (DSPI_RXFR6) 0xFC05_C0B4 (DSPI_RXFR14)
0xFC05_C098 (DSPI_RXFR7) 0xFC05_C0B8 (DSPI_RXFR15)

Access: User read-only

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

Figure 33-10. DSPI Receive FIFO Registers (DSPI_RXFR*n*)

Table 33-11. DSPI_RXFR*n* Field Description

Field	Description
31–16	Reserved, must be cleared.
15–0 RXDATA	Receive data. Contains the received SPI data.

33.4 Functional Description

The DSPI supports full-duplex, synchronous serial communications between the MCU and external peripheral devices. The DSPI supports up to 32 queued SPI transfers at once (16 transmit and 16 receive) in the DSPI resident FIFOs, thereby eliminating CPU intervention between transfers.

The DSPI_CTAR*n* registers hold clock and transfer attributes. The SPI configuration can select which CTAR to use on a frame by frame basis by setting the DSPI_PUSHR[CTAS] field. See Section 33.3.3, "DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTARn)," for information on DSPI_CTAR*n* fields.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT and SIN signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data exchanged between the master and the slave; the data that was in the master's shift register is now in the shift register of the slave and vice versa. At the end of a transfer, the DSPI_SR[TCF] bit is set to indicate a completed transfer. Figure 33-11 illustrates how master and slave data is exchanged.

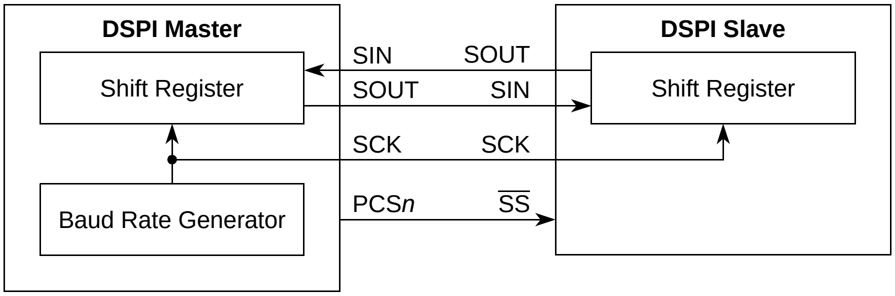


Figure 33-11. SPI Serial Protocol Overview

The DSPI has four peripheral chip select (DSPI_PCS*n*) signals that select which of the slaves to communicate with.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in Section 33.4.4, "Transfer Formats." The transfer rate and delay settings are described in section Section 33.4.3, "DSPI Baud Rate and Clock Delay Generation."

See Section 33.4.7, "Power Saving Features" for information on the power-saving features of the DSPI.

33.4.1 Start and Stop of DSPI Transfers

The DSPI has two operating states; stopped and running. The default state of the DSPI is stopped. In the stopped state, no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The stopped state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. Master/slave mode must only be changed when the DSPI is halted (DSPI_MCR[HALT] is set). The DSPI_SR[TXRXS] bit is cleared in this state. In the running state, serial transfers take place. The DSPI_SR[TXRXS] bit is set in the running state. Figure 33-12 shows a state diagram of the start and stop mechanism. The transitions are described in Table 33-12.

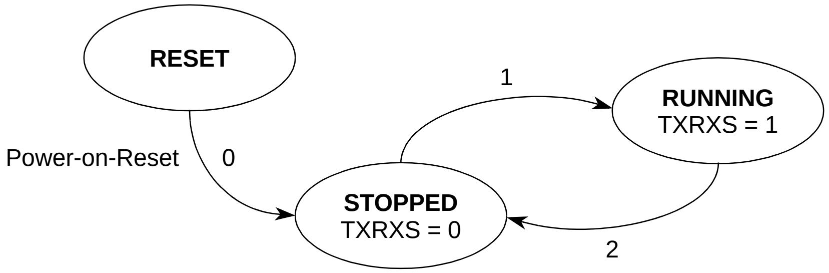


Figure 33-12. DSPI Start and Stop State Diagram

Table 33-12. State Transitions for Start and Stop of DSPI Transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI is started (DSPI transitions to running) when all of the following conditions are true: • EOQF bit is clear • Debug mode is unselected or the FRZ bit is clear • HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from running to stopped) after the current frame for any one of the following conditions: • EOQF bit is set • Debug mode is selected and the FRZ bit is set • HALT bit is set

State transitions from running to stopped occur on the next frame boundary if a transfer is in progress or on the next system clock cycle if no transfers are in progress.

33.4.2 Serial Peripheral Interface (SPI) Configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The SPI frames can be from 4–16 bits long. The data transmitted can come from queues stored in RAM external to the DSPI. Host software or the eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or the eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI. The FIFO buffer operations are described in Section 33.4.2.4, “TX FIFO Buffering Mechanism,” and Section 33.4.2.5, “RX FIFO Buffering Mechanism.” The interrupt and DMA request conditions are described in Section 33.4.6, “Interrupts/DMA Requests.”

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for both modes. In master mode, the DSPI initiates and controls the transfer according to the SPI command field of the TX FIFO entry. In slave mode, the DSPI only responds to transfers initiated by a bus master external to the DSPI, and the SPI command field of the TX FIFO entry is ignored. For information on switching between master and slave modes see Section 33.5.2, “Switching Master and Slave Mode.”

33.4.2.1 Master Mode

In master mode, the DSPI initiates the serial transfers by controlling the serial communications clock (DSPI_SCK) and the peripheral chip select (DSPI_PCS*n*) signals. The SPI command field in the executing TX FIFO entry determines which DSPI_CTAR*n* register sets the transfer attributes and which DSPI_PCS*n* signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See Section 33.3.6, "DSPI Push Transmit FIFO Register (DSPI_PUSHR)," for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (DSPI_SOUT) pin. In master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

33.4.2.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase, and frame size must be set for successful communication with an SPI master. The slave mode transfer attributes are set in the DSPI_CTAR0 register.

33.4.2.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX or RX FIFOs. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The FIFOs are disabled separately; setting the DSPI_MCR[DIS_TXF] bit disables the TX FIFO, and setting the DSPI_MCR[DIS_RXF] bit disables the RX FIFO.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR. When the TX FIFO is disabled, DSPI_SR[TFFF, TFUF, and TXCTR] fields behave as if there is a one-entry FIFO, but the contents of DSPI_TXFRs and TXNXTPTR are undefined. Likewise, when RX FIFO is disabled, DSPI_SR[RFDF, RFOF, and RXCTR] fields behave as if there is a one-entry FIFO, but the contents of DSPI_RXFRs and POPNXTPTR are undefined.

The TX and RX FIFOs should be disabled only if the application's operating mode requires FIFO to be disabled. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported and may result in incorrect results.

NOTE

When the FIFOs are disabled, the respective DSPI_MCR[CLR_TXF, CLR_RXF] bits have no effect.

33.4.2.4 TX FIFO Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds 16 entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI_PUSHR). For more information on DSPI_PUSHR, refer to Section 33.3.6, "DSPI Push Transmit FIFO Register (DSPI_PUSHR)." TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPI_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data transfers into the shift register from the TX FIFO. For more information on DSPI_SR, refer to Section 33.3.4, “DSPI Status Register (DSPI_SR).”

The DSPI_SR[TXNXTPTR] field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means DSPI_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field increments every time SPI data transfers from TX FIFO to shift register.

33.4.2.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR register. When the TX FIFO is not full, the TX FIFO fill flag, DSPI_SR[TFFF], is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI_PUSHR is complete. Host software writing a 1 to the DSPI_SR[TFFF] bit can also clear the TFFF bit. The TFFF can generate a DMA request or an interrupt request. See Section 33.4.6.2, “Transmit FIFO Fill Interrupt or DMA Request (TFFF),” for details.

The DSPI ignores attempts to push data to a full TX FIFO; in other words, the state of the TX FIFO is unchanged and no error condition is indicated.

33.4.2.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter decrements by one. At the end of a transfer, the DSPI_SR[TCF] bit is set to indicate completion of a transfer. The TX FIFO is flushed by writing a 1 to the DSPI_MCR[CLR_TXF] bit.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the slave’s transmit FIFO underflow flag, DSPI_SR[TFUF], is set. See Section 33.4.6.4, “Transmit FIFO Underflow Interrupt Request (TFUF),”for details.

33.4.2.5 RX FIFO Buffering Mechanism

The RX FIFO functions as a buffer for data received on the DSPI_SIN pin. The RX FIFO holds 16 received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPI_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO. For more information on the DSPI_POPR, refer to Section 33.3.7, “DSPI Pop Receive FIFO Register (DSPI_POPR).”

The RX FIFO counter field, DSPI_SR[RXCTR], indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The DSPI_SR[POPNXTPTR] field points to the RX FIFO entry returned when the DSPI_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPI_RXFR0. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that is returned when DSPI_POPR is read. The POPNXTPTR field increments every time the DSPI_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

33.4.2.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO, the RX FIFO counter increments by one.

If the RX FIFO and shift register are full and a transfer is initiated, the DSPI_SR[RFOF] bit is asserted indicating an overflow condition. Depending on the state of the DSPI_MCR[ROOE] bit, data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is set, incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

33.4.2.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI_POPR. For more information on DSPI_POPR, refer to Section 33.3.7, “DSPI Pop Receive FIFO Register (DSPI_POPR).” A read of the DSPI_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, and the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

When the RX FIFO is not empty, the RX FIFO drain flag, DSPI_SR[RFDF], is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPI_POPR is complete. Alternatively, the RFDF bit can be cleared by software writing a 1 to it.

33.4.3 DSPI Baud Rate and Clock Delay Generation

The DSPI_SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate. Figure 33-13 shows conceptually how the DSPI_SCK signal is generated.

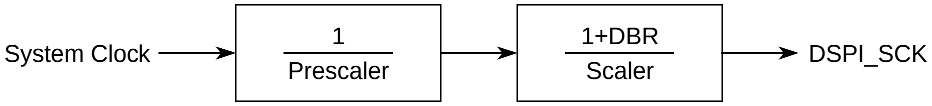


Figure 33-13. Communications Clock Prescalers and Scalers

33.4.3.1 Baud Rate Generator

The baud rate is the frequency of the serial communication clock (DSPI_SCK). The system clock is divided by a baud rate prescaler (defined by DSPI_CTAR*n*[PBR]) and baud rate scaler (defined by DSPI_CTAR*n*[BR]) to produce DSPI_SCK with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI_CTAR*n* select the frequency of DSPI_SCK using the following formula:

$$\text{SCK baud rate} = \frac{f_{SYS/3}}{\text{PBR Prescaler Value}} \times \frac{1 + \text{DBR}}{\text{BR Scaler Value}} \qquad \textbf{\textit{Eqn. 33-1}}$$

Table 33-13 shows an example of a computed baud rate.

Table 33-13. Baud Rate Computation Example

$f_{SYS/3}$	PBR	Prescaler Value	BR	Scaler Value	DBR Value	Baud Rate
100 MHz	00	2	0000	2	0	25 Mb/s
20 MHz	00	2	0000	2	1	10 Mb/s

33.4.3.2 PCS to SCK Delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of DSPI_PCS signal to the first DSPI_SCK edge. See Figure 33-14 for an illustration of the PCS to SCK delay. The DSPI_CTAR*n*[PCSSCK, CSSCK] fields select the PCS to SCK delay, and the relationship is expressed by the following:

$$t_{CSC} = \frac{1}{f_{SYS/3}} \times PCSSCK \times CSSCK \qquad \textit{Eqn. 33-2}$$

Table 33-14 shows an example of the computed PCS to SCK delay.

Table 33-14. PCS to SCK Delay Computation Example

PCSSCK	Prescaler Value	CSSCK	Scaler Value	$f_{SYS/3}$	PCS to SCK Delay
01	3	0100	32	100 MHz	0.96 μs

33.4.3.3 After SCK Delay (t_{ASC})

The after SCK delay is the length of time between the last edge of DSPI_SCK and negation of DSPI_PCS. See Figure 33-14 and Figure 33-15 for illustrations of the after SCK delay. The DSPI_CTAR*n*[PASC, ASC] fields select the after SCK delay. The relationship between these variables is given in the following:

$$t_{ASC} = \frac{1}{f_{SYS/3}} \times PASC \times ASC \qquad \textit{Eqn. 33-3}$$

Table 33-15 shows an example of the computed after SCK delay.

Table 33-15. After SCK Delay Computation Example

PASC	Prescaler Value	ASC	Scaler Value	$f_{SYS/3}$	After SCK Delay
01	3	0100	32	100 MHz	0.96 us

33.4.3.4 Delay after Transfer (t_{DT})

The delay after transfer is the length of time between negation of DSPI_PCS signal for a frame and the assertion of DSPI_PCS signal for the next frame. See Figure 33-14 for an illustration of the delay after transfer. DSPI_CTAR*n*[PDT, DT] fields select the delay after transfer by the formula:

$$t_{DT} = \frac{1}{f_{SYS/3}} \times PDT \times DT \qquad \textit{Eqn. 33-4}$$

Table 33-16 shows an example of the computed delay after transfer.

Table 33-16. Delay after Transfer Computation Example

PDT	Prescaler Value	DT	Scaler Value	$f_{SYS/3}$	Delay after Transfer
01	3	1110	32768	100 MHz	0.98 ms

33.4.4 Transfer Formats

The serial communications clock (DSPI_SCK) signal and the DSPI_PCS*n* signals control the SPI serial communication. The DSPI_SCK signal provided by the master device synchronizes shifting and sampling of the data by the DSPI_SIN and DSPI_SOUT pins. The DSPI_PCS*n* signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the DSPI_CTAR*n*[CPOL, CPHA] bits select the polarity and phase of the DSPI_SCK signal. The polarity bit selects the idle state of the DSPI_SCK. The clock phase bit selects if the data on DSPI_SOUT is valid before or on the first DSPI_SCK edge.

When the DSPI is the bus slave, the DSPI_CTAR0[CPOL, CPHA] bits select the polarity and phase of the serial clock. Even though the bus slave does not control the DSPI_SCK signal, clock polarity, clock phase, and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The DSPI_MCR[MTFE] bit selects between classic SPI format and modified transfer format. The classic SPI formats are described in Section 33.4.4.1, “Classic SPI Transfer Format (CPHA = 0)” and Section 33.4.4.2, “Classic SPI Transfer Format (CPHA = 1).” The modified transfer formats are described in Section 33.4.4.3, “Modified SPI Transfer Format (MTFE = 1, CPHA = 0)” and Section 33.4.4.4, “Modified SPI Transfer Format (MTFE = 1, CPHA = 1).”

33.4.4.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in Figure 33-14 communicates with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their DSPI_SIN pins on the odd-numbered DSPI_SCK edges and change the data on their DSPI_SOUT pins on the even-numbered DSPI_SCK edges.

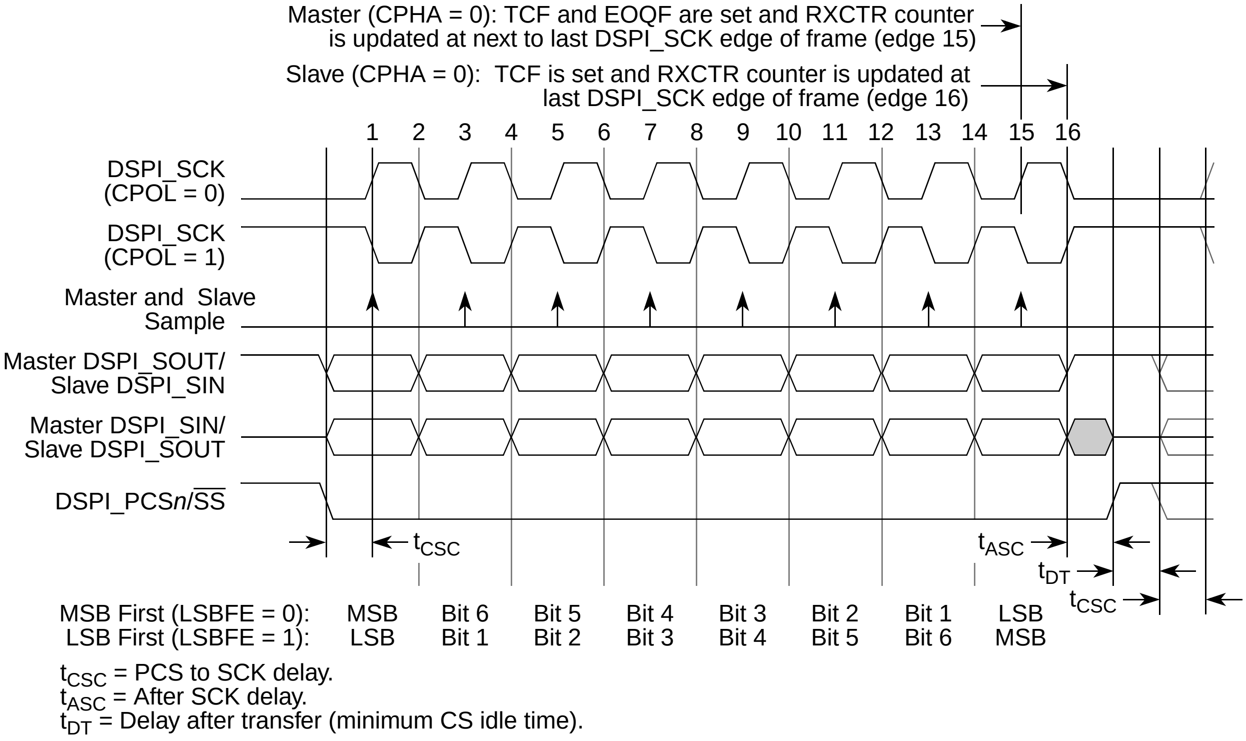

Master (CPHA = 0): TCF and EOQF are set and RXCTR counter is updated at next to last DSPI_SCK edge of frame (edge 15)
Slave (CPHA = 0): TCF is set and RXCTR counter is updated at last DSPI_SCK edge of frame (edge 16)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
DSPI_SCK (CPOL = 0)
DSPI_SCK (CPOL = 1)
Master and Slave Sample
Master DSPI_SOUT/ Slave DSPI_SIN
Master DSPI_SIN/ Slave DSPI_SOUT
DSPI_PCSn/SS
tCSC
tASC
tDT
tCSC
MSB First (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB First (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB
tCSC = PCS to SCK delay.
tASC = After SCK delay.
tDT = Delay after transfer (minimum CS idle time).

Figure 33-14. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

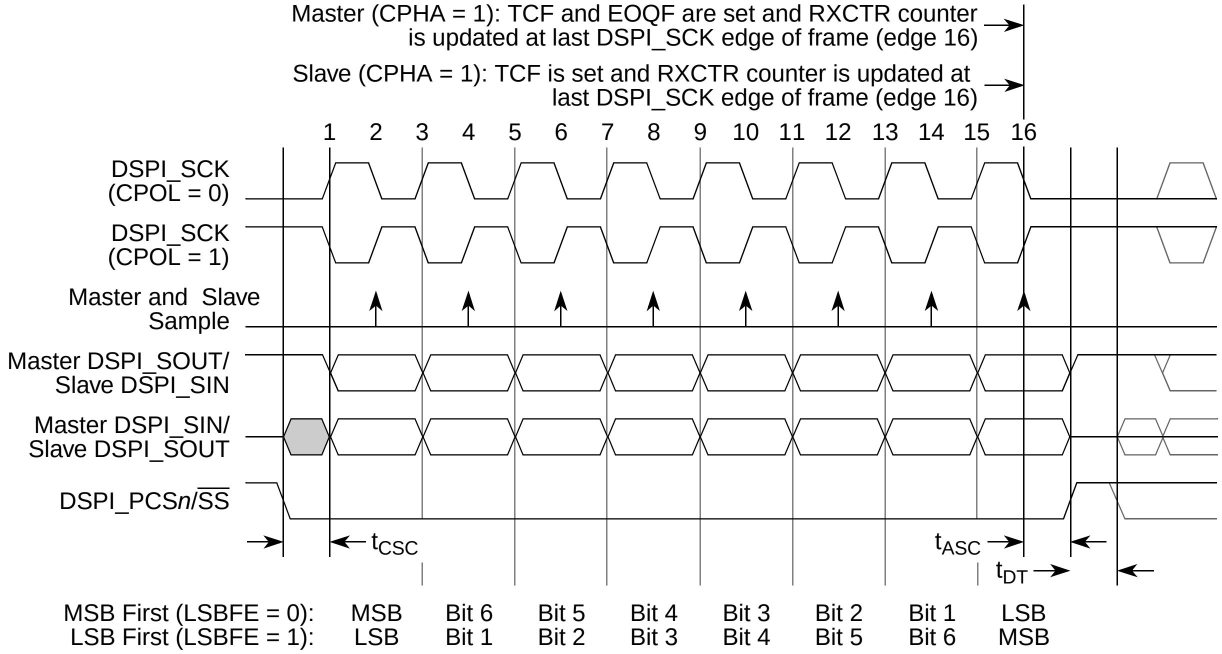
The master initiates the transfer by placing its first data bit on the DSPI_SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its DSPI_SOUT pin. After the t_{CSC} delay elapses, the master outputs the first edge of DSPI_SCK. The master and slave devices use this edge to sample the first input data bit on their serial data input signals. At the second edge of the DSPI_SCK, the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame, the master and the slave sample their DSPI_SIN pins on the odd-numbered clock edges and change the data on their DSPI_SOUT pins on the even-numbered clock edges. After the last clock edge occurs, a delay of t_{ASC} is inserted before the master negates the DSPI_PCS*n* signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

If DSPI_CTAR*n*[CPHA] is cleared:

- At the next to last serial clock edge of the frame (edge 15 of Figure 33-14)
  - — Master's TCF and EOQF are set and RXCTR counter is updated
- At the last serial clock edge of the frame (edge 16 of Figure 33-14)
  - — Slave's TCF is set and RXCTR counter is updated

33.4.4.2 Classic SPI Transfer Format (CPHA = 1)

The transfer format shown in Figure 33-15 communicates with peripheral SPI slave devices that require the first DSPI_SCK edge before the first data bit becomes available on the slave DSPI_SOUT pin. In this format, the master and slave devices change the data on their DSPI_SOUT pins on the odd-numbered DSPI_SCK edges and sample the data on their DSPI_SIN pins on the even-numbered DSPI_SCK edges.


Master (CPHA = 1): TCF and EOQF are set and RXCTR counter is updated at last DSPI_SCK edge of frame (edge 16)
Slave (CPHA = 1): TCF is set and RXCTR counter is updated at last DSPI_SCK edge of frame (edge 16)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
DSPI_SCK (CPOL = 0)
DSPI_SCK (CPOL = 1)
Master and Slave Sample
Master DSPI_SOUT/ Slave DSPI_SIN
Master DSPI_SIN/ Slave DSPI_SOUT
DSPI_PCSn/SS
tCSC
tASC
tDT
MSB First (LSBFE = 0): MSB Bit 6 Bit 5 Bit 4 Bit 3 Bit 2 Bit 1 LSB
LSB First (LSBFE = 1): LSB Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 MSB

t_{CSC} = PCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 33-15. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the DSPI_PCS*n* signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first DSPI_SCK edge and places valid data on the master DSPI_SOUT pin. The slave responds to the first DSPI_SCK edge by placing its first data bit on its slave DSPI_SOUT pin.

At the second edge of the DSPI_SCK, the master and slave sample their DSPI_SIN pins. For the rest of the frame, the master and the slave change the data on their DSPI_SOUT pins on the odd-numbered clock edges and sample their DSPI_SIN pins on the even-numbered clock edges. After the last clock edge occurs,1 a delay of t_{ASC} is inserted before the master negates the DSPI_PCS*n* signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

If DSPI_CTAR*n*[CPHA] is set:

- At the last serial clock edge (edge 16 of Figure 33-15)
  - — Master’s EOQF and TCF are set
  - — Slave’s TCF is set
  - — Master’s and slave’s RXCTR counters are updated

33.4.4.3 Modified SPI Transfer Format (MTFE = 1, CPHA = 0)

In this modified transfer format, the master and the slave sample later in the DSPI_SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the DSPI_SCK period as the DSPI_SCK period decreases with increasing baud rates.

NOTE

For correct operation of the modified transfer format, thoroughly analyze the SPI link timing budget.

The master and the slave place data on the DSPI_SOUT pins at the assertion of the DSPI_PCS*n* signal. After the PCS to SCK delay has elapsed, the first DSPI_SCK edge is generated. The slave samples the master DSPI_SOUT signal on every odd numbered DSPI_SCK edge. The slave also places new data on the slave DSPI_SOUT on every odd numbered clock edge.

The master places its second data bit on the DSPI_SOUT line one system clock after odd numbered SDSPI_CK edge. Writing to the DSPI_MCR[SMPL_PT] field selects the point where the master samples the slave DSPI_SOUT. Table 33-17 lists the number of system clock cycles between the active edge of DSPI_SCK and the master sample point for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 33-17. Delayed Master Sample Point

SMPL_PT	Number of System Clock Cycles between Odd-numbered Edge of SCK and Sampling of SIN
00	0
01	1
10	2
11	Reserved

Figure 33-16 shows the modified transfer format for CPHA is cleared. Only the condition where CPOL is cleared is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

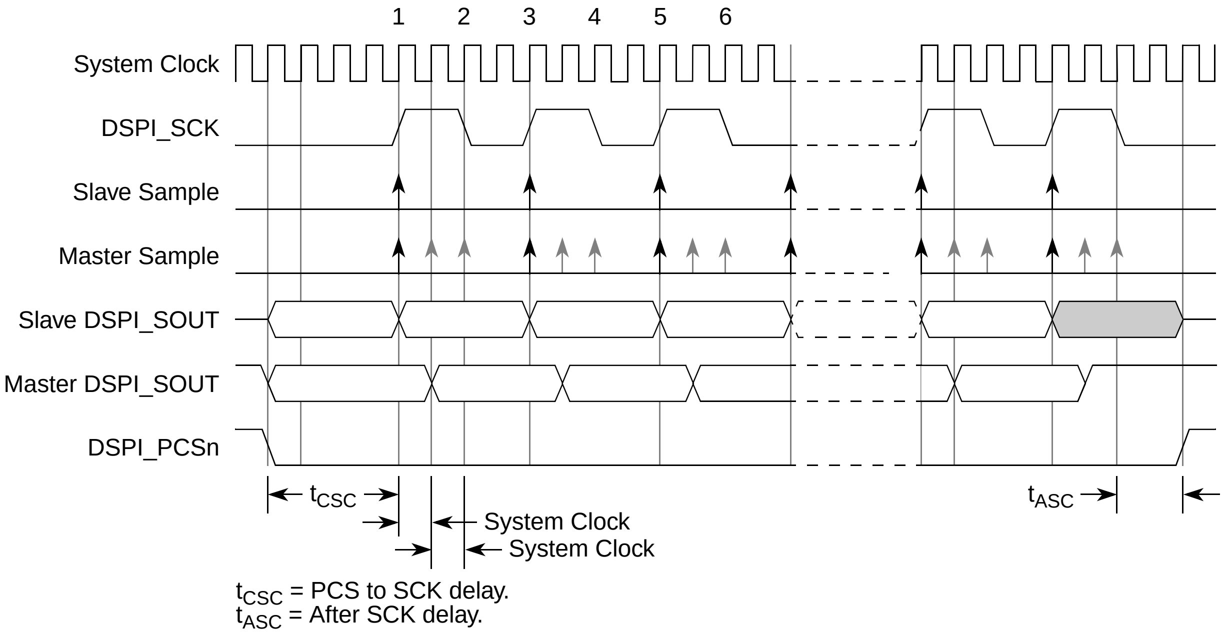


Figure 33-16. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, Fsck = Fsys/4)

33.4.4.4 Modified SPI Transfer Format (MTFE = 1, CPHA = 1)

Figure 33-17 shows the modified transfer format for CPHA is set. Only the condition where CPOL is cleared is described. At the start of a transfer, the DSPI asserts the DSPI_PCS*n* signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their DSPI_SOUT pins at the first edge of DSPI_SCK. The slave samples the master DSPI_SOUT signal on the even numbered edges of DSPI_SCK. The master samples the slave DSPI_SOUT signal on the odd numbered DSPI_SCK edges starting with the third DSPI_SCK edge. The slave samples the last bit on the last edge of the DSPI_SCK. The master samples the last slave DSPI_SOUT bit one half DSPI_SCK cycle after the last edge of DSPI_SCK. No clock edge is visible on the master DSPI_SCK pin during the sampling of the last bit. The SCK to PCS delay must be greater or equal to half of the DSPI_SCK period.

NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

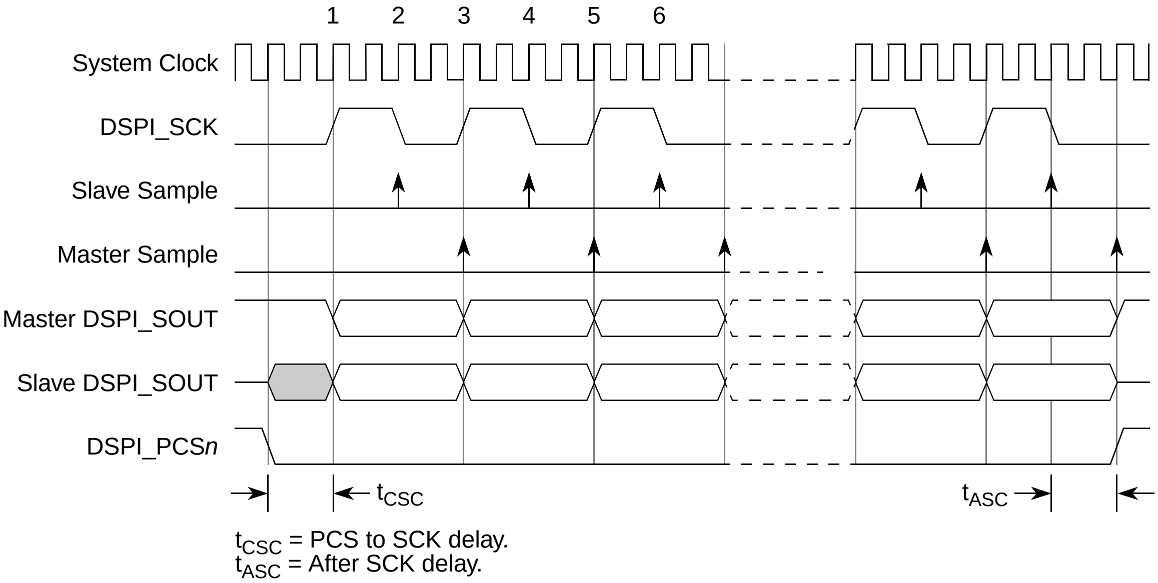


Figure 33-17. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1, Fsck = Fsys/4)

33.4.4.5 Continuous Selection Format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the DSPI_PUSHR[CONT] bit.

When CONT is cleared, DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the DSPI_MCR[PCSIS] field. Figure 33-18 shows the timing diagram for two four-bit transfers with CPHA set and CONT cleared.

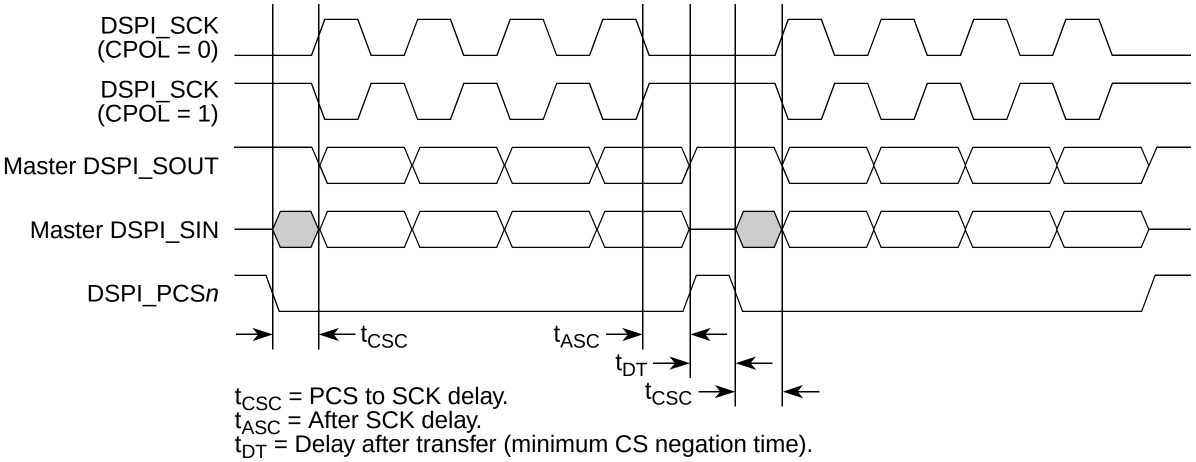

DSPI_SCK (CPOL = 0)
DSPI_SCK (CPOL = 1)
Master DSPI_SOUT
Master DSPI_SIN
DSPI_PCSn
t_{CSC}
t_{ASC}
t_{DT}
t_{CSC}
t_{CSC} = PCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 33-18. Example of Non-Continuous Format (CPHA=1, CONT=0)

When CONT is set and the DSPI_PCS*n* signal for the next transfer the same as for the current transfer, DSPI_PCS*n* signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers. Figure 33-19 shows the timing diagram for two four-bit transfers with CPHA and CONT set.

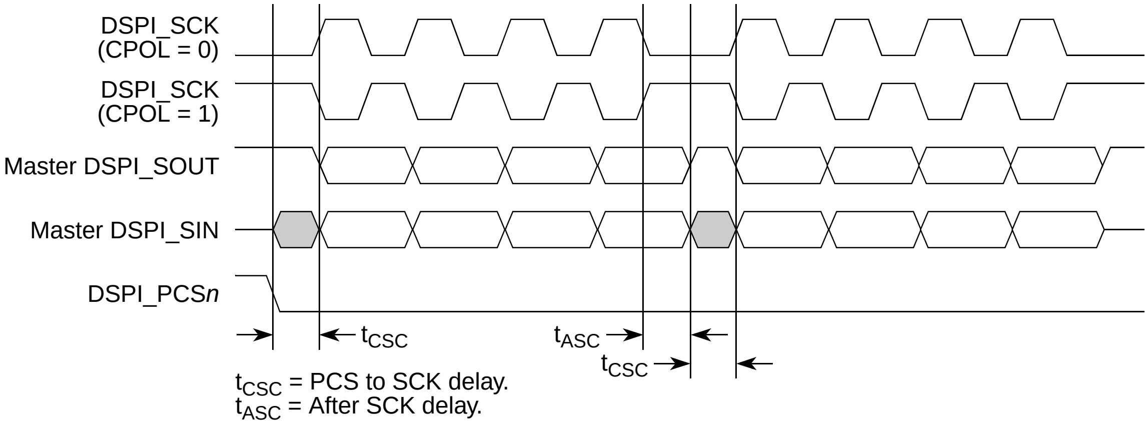


Figure 33-19. Example of Continuous Transfer (CPHA = 1, CONT = 1)

In Figure 33-19, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}. It does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching DSPI_CTAR*n* registers between frames while using continuous selection can cause errors in the transfer. The DSPI_PCS*n* signal must be negated before DSPI_CTAR is switched.

When CONT is set and the DSPI_PCS*n* signals for the next transfer are different from the present transfer, the DSPI_PCS*n* signals behave as if the CONT bit was cleared.

33.4.4.6 Clock Polarity Switching between DSPI Transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame. In Figure 33-20, time A shows the one clock interval. Time B is user programmable from a minimum of 2 system clocks. See Section 33.3.3, “DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTARn).”

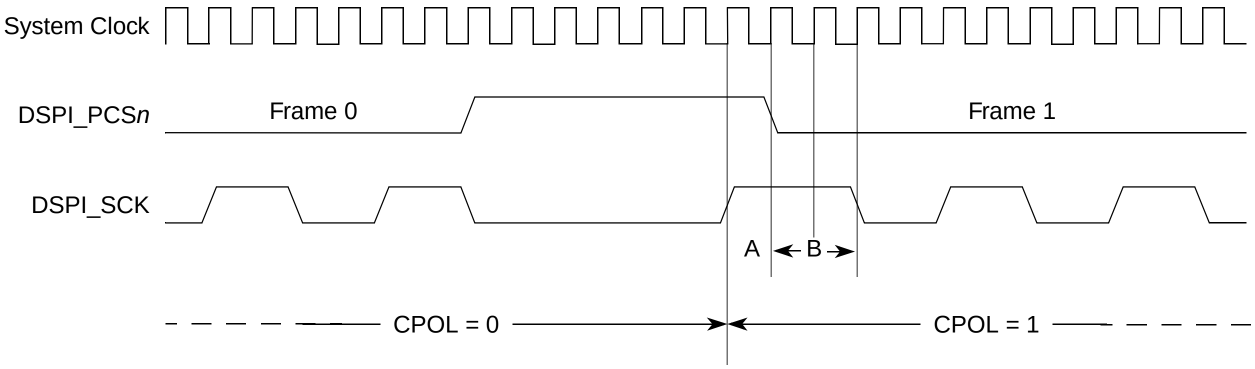


Figure 33-20. Polarity Switching between Frames

33.4.5 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous DSPI_SCK signal for slave peripherals that require a continuous clock. Continuous SCK is enabled by setting the DSPI_MCR[CONT_SCKE] bit.

Continuous SCK is only supported if CPHA is set. Clearing CPHA is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- DSPI_CTAR0 is used initially. At the start of each SPI frame transfer, the DSPI_CTAR*n* specified by the CTAS field for the frame is used.
- The currently selected DSPI_CTAR*n* remains in use until the start of a frame with a different DSPI_CTAR*n* specified, or the continuous SCK mode is terminated.

It is recommended that the baud rate is the same for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the PCS to SCK delay and the after SCK delay. The delay after transfer is fixed at one DSPI_SCK cycle. Figure 33-21 shows timing diagram for continuous SCK format with continuous selection disabled.

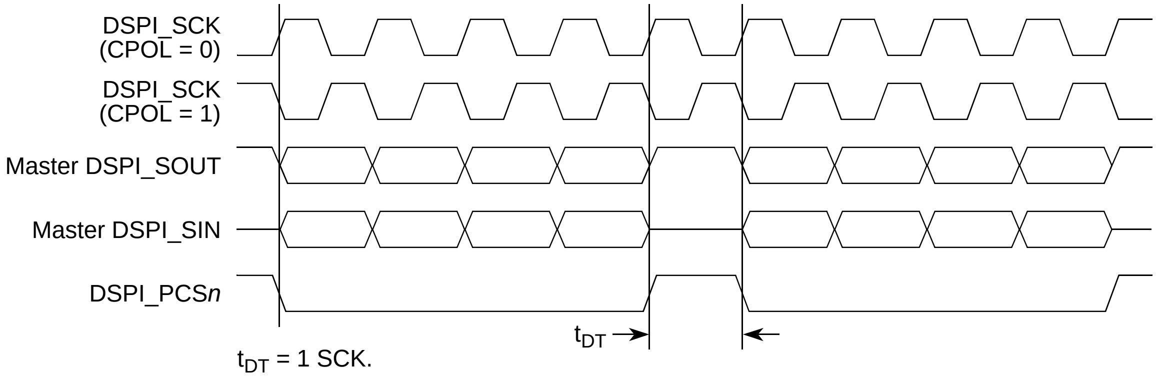


Figure 33-21. Continuous SCK Timing Diagram (CONT= 0)

If the CONT bit in the TX FIFO entry is set, DSPI_PCS*n* remains asserted between the transfers when the DSPI_PCS*n* signal for the next transfer is the same as for the current transfer. Figure 33-22 shows timing diagram for continuous SCK format with continuous selection enabled.

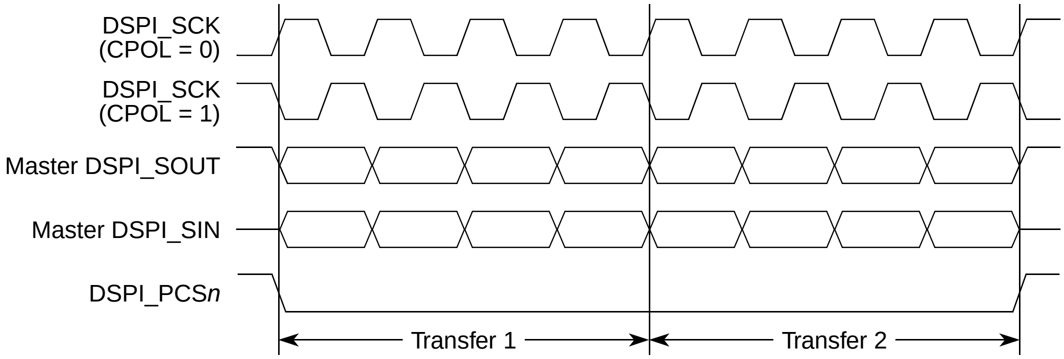


Figure 33-22. Continuous SCK Timing Diagram (CONT=1)

33.4.6 Interrupts/DMA Requests

The DSPI has six conditions that can only generate interrupt requests and two conditions that can generate an interrupt or DMA request. Table 33-18 lists these conditions.

Table 33-18. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	—
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	—
TX FIFO underflow has occurred	TFUF	X	—
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow has occurred	RFOF	X	—

Table 33-18. Interrupt and DMA Request Conditions (continued)

Condition	Flag	Interrupt	DMA
A FIFO overrun has occurred[1]	TFUF OR RFOF	X	—
General DSPI condition[2]	Any of the above	X	—

[1] The FIFO overrun condition is created by OR-ing the TFUF and RFOF flags together.

[2] OR'd condition of any of the six flags.

Each condition has a flag bit and a request enable bit. The flag bits are described in Section 33.3.4, "DSPI Status Register (DSPI_SR)," and the request enable bits are described in Section 33.3.5, "DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)." The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the DSPI_RSER[TFFF_DIRS, RFDF_DIRS] bits.

33.4.6.1 End of Queue Interrupt Request (EOQF)

The end of queue equest indicates end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the DSPI_RSER[EOQF_RE] bit is set. See the EOQ bit description in Section 33.3.4, "DSPI Status Register (DSPI_SR)." Refer to Figure 33-14 and Figure 33-15 that illustrate when EOQF is set.

33.4.6.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the DSPI_RSER[TFFF_RE] bit is set. The DSPI_RSER[TFFF_DIRS] bit selects whether a DMA request or an interrupt request is generated.

33.4.6.3 Transfer Complete Interrupt Request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the DSPI_RSER[TCF_RE] bit is set. See the TCF bit description in Section 33.3.4, "DSPI Status Register (DSPI_SR)." Refer to Figure 33-14 and Figure 33-15 that illustrate when TCF is set.

33.4.6.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the DSPI_RSER[TFUF_RE] bit is set, an interrupt request is generated.

33.4.6.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the DSPI_RSER[RFDF_RE] bit is set. The DSPI_RSER[RFDF_DIRS] bit selects whether a DMA request or an interrupt request is generated.

33.4.6.6 Receive FIFO Overflow Interrupt Request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The DSPI_RSER[RFOF_RE] bit must be set for the interrupt request to be generated.

Depending on the state of the DSPI_MCR[ROOE] bit, data from the transfer that generated overflow is ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, incoming data is ignored.

33.4.6.7 FIFO Overrun Request (TFUF) or (RFOF)

The FIFO overrun request indicates at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR’ing the RX FIFO overflow and TX FIFO underflow signals.

33.4.7 Power Saving Features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

33.4.7.1 Module Disable Mode

Module disable mode is a mode the DSPI can enter to save power. Host software can initiate the module disable mode by setting DSPI_MCR[MDIS]. The MDIS bit is set at reset.

In module disable mode, the DSPI is in a dormant state, but the memory-mapped registers remain accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any affect in module disable mode. Changes to the DSPI_MCR[DIS_TXF, DIS_RXF] fields do not have any affect in module disable mode. In module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no effect. Writing to the DSPI_TCR during module disable mode does not have any affect. Interrupt and DMA request signals cannot be cleared while in module disable mode.

33.4.7.2 Slave Interface Signal Gating

The DSPI’s module enable signal gates slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

33.5 Initialization/Application Information

33.5.1 How to Change Queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag, DSPI_SR[EOQF] is set.
3. The setting of the EOQF flag disables serial transmission and serial reception of data, putting the DSPI in the stopped state. The TXRXS bit is cleared to indicate the stopped state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the DSPI_SR[RXCNT] bit or by checking the DSPI_SR[RFDF] bit after each read operation of the DSPI_POPR register.
7. Modify DMA descriptor of TX and RX channels for new queues.
8. Flush TX FIFO by writing a 1 to the DSPI_MCR[CLR_TXF] bit; Flush RX FIFO by writing a 1 to the DSPI_MCR[CLR_RXF] bit.
9. Clear transfer count by setting the CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to the DSPI_TCR[SPI_TCNT] field.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

33.5.2 Switching Master and Slave Mode

When changing modes in the DSPI, follow the steps below to guarantee proper operation.

1. Halt the DSPI by setting DSPI_MCR[HALT].
2. Clear the transmit and receive FIFOs by writing a 1 to the CLR_TXF and CLR_RXF bits in DSPI_MCR.
3. Set the appropriate mode in DSPI_MCR[MSTR] and enable the DSPI by clearing DSPI_MCR[HALT].

33.5.3 Baud Rate Settings

Table 33-19 shows the baud rate generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI_CTAR*n* registers. The values calculated assume a 100 MHz system frequency.

Table 33-19. Baud Rate Values

		Baud Rate Divider Prescaler Values (DSPI_CTAR*n*[PBR])			
		2	**3**	**5**	**7**
Baud Rate Scaler Values (DSPI_CTAR*n*[BR])	**2**	25.0MHz	16.7MHz	10.0MHz	7.14MHz
	4	12.5MHz	8.33MHz	5.00MHz	3.57MHz
	6	8.33MHz	5.56MHz	3.33MHz	2.38MHz
	8	6.25MHz	4.17MHz	2.50MHz	1.79MHz
	16	3.12MHz	2.08MHz	1.25MHz	893kHz
	32	1.56MHz	1.04MHz	625kHz	446kHz
	64	781kHz	521kHz	312kHz	223kHz
	128	391kHz	260kHz	156kHz	112kHz
	256	195kHz	130kHz	78.1kHz	55.8kHz
	512	97.7kHz	65.1kHz	39.1kHz	27.9kHz
	1024	48.8kHz	32.6kHz	19.5kHz	14.0kHz
	2048	24.4kHz	16.3kHz	9.77kHz	6.98kHz
	4096	12.2kHz	8.14kHz	4.88kHz	3.49kHz
	8192	6.10kHz	4.07kHz	2.44kHz	1.74kHz
	16384	3.05kHz	2.04kHz	1.22kHz	872Hz
	32768	1.53kHz	1.02kHz	610Hz	436Hz

33.5.4 Delay Settings

Table 33-20 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR*n* registers. The values calculated assume a 100 MHz system frequency.

Table 33-20. Delay Values

		Delay Prescaler Values (DSPI_CTAR*n*[PBR])			
		1	**3**	**5**	**7**
Delay Scaler Values (DSPI_CTAR*n*[DT])	**2**	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 μs
	32	320.0 ns	960.0 ns	1.6 μs	2.2 μs
	64	640.0 ns	1.9 μs	3.2 μs	4.5 μs
	128	1.3 μs	3.8 μs	6.4 μs	9.0 μs
	256	2.6 μs	7.7 μs	12.8 μs	17.9 μs
	512	5.1 μs	15.4 μs	25.6 μs	35.8 μs
	1024	10.2 μs	30.7 μs	51.2 μs	71.7 μs
	2048	20.5 μs	61.4 μs	102.4 μs	143.4 μs
	4096	41.0 μs	122.9 μs	204.8 μs	286.7 μs
	8192	81.9 μs	245.8 μs	409.6 μs	573.4 μs
	16384	163.8 μs	491.5 μs	819.2 μs	1.1 ms
	32768	327.7 μs	983.0 μs	1.6 ms	2.3 ms
	65536	655.4 μs	2.0 ms	3.3 ms	4.6 ms

33.5.5 Calculation of FIFO Pointer Addresses

Complete visibility of the TX and RX FIFO contents is available through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO, the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO, the first-in pointer is the pop next pointer (POPNXTPTR).

Figure 33-23 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See Section 33.4.2.4, “TX FIFO Buffering Mechanism,” and Section 33.4.2.5, “RX FIFO Buffering Mechanism,” for details on the FIFO operation.

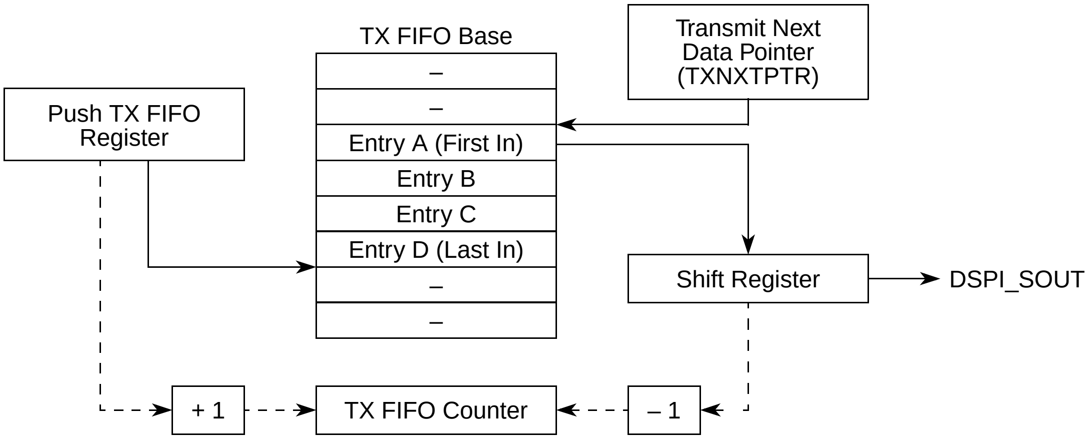


Figure 33-23. TX FIFO Pointers and Counter

33.5.5.1 Address Calculation for the First-in and Last-in Entries in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

First-in entry address = TXFIFO base + 4 × (TXNXTPTR)

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Last-in entry address = TX FIFO base + 4 × [(TXCTR + TXNXTPTR - 1) modulo TX FIFO depth]

where:

TX FIFO base: base address of TX FIFO

TXCTR: TX FIFO counter

TXNXTPTR: transmit next pointer

TX FIFO depth: 16

33.5.5.2 Address Calculation for the First-in and Last-in Entries in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

First-in entry address = RX FIFO base + 4 × (POPNXTPTR)

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Last-in entry address = RX FIFO base + 4 × [(RXCTR + POPNXTPTR - 1) modulo RX FIFO depth]

RX FIFO base: base address of RX FIFO

RXCTR: RX FIFO counter

POPNXTPTR: pop next pointer

RX FIFO depth: 16

Chapter 34
UART Modules

34.1 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.

34.1.1 Overview

The internal bus clock can clock each of the three independent UARTs, eliminating the need for an external UART clock. As Figure 34-1 shows, each UART module interfaces directly to the CPU and consists of:

- Serial communication channel
- Programmable clock generation
- Interrupt control logic and DMA request logic
- Internal channel control logic

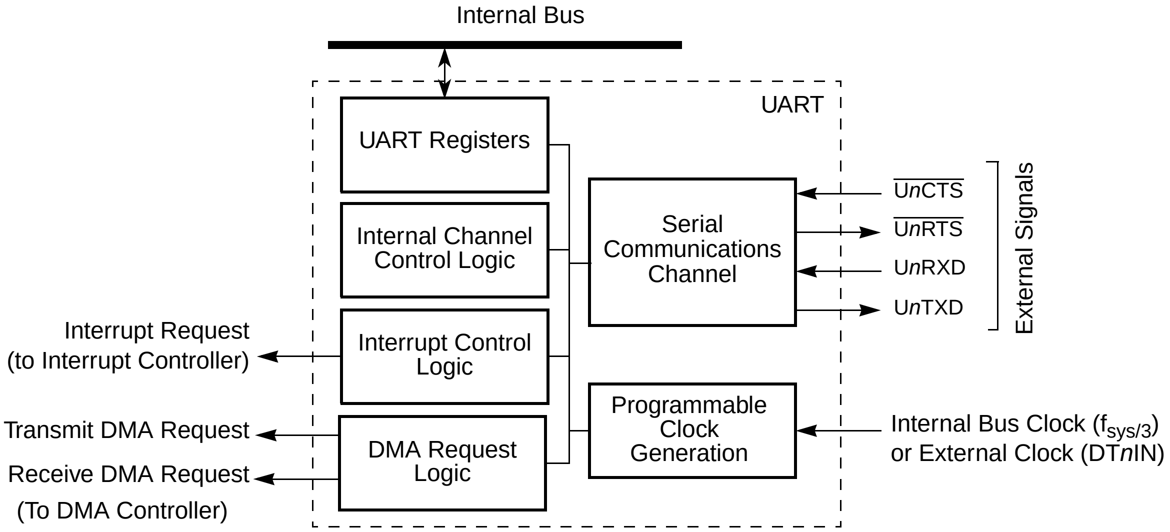

Internal Bus
UART
UART Registers
Internal Channel Control Logic
Interrupt Control Logic
DMA Request Logic
Serial Communications Channel
Programmable Clock Generation
UnCTS
UnRTS
UnRXD
UnTXD
External Signals
Interrupt Request (to Interrupt Controller)
Transmit DMA Request
Receive DMA Request
(To DMA Controller)
Internal Bus Clock ($f_{sys/3}$) or External Clock (DTnIN)

Figure 34-1. 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 34.4.2.1, “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 34.4.2.2, “Receiver.”

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to Chapter 14, “Pin-Multiplexing and Control”) prior to configuring the UART module.

34.1.2 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

34.2 External Signal Description

Table 34-1 briefly describes the UART module signals.

Table 34-1. UART Module External Signals

Signal	Description
U*n*TXD	Transmitter Serial Data Output. U*n*TXD is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on U*n*TXD on the falling edge of the clock source, with the least significant bit (lsb) sent first.
U*n*RXD	Receiver Serial Data Input. Data received on U*n*RXD is sampled on the rising edge of the clock source, with the lsb received first.
$\overline{\mathrm{U}n\mathrm{CTS}}$	Clear-to- Send. This input can generate an interrupt on a change of state.
$\overline{\mathrm{U}n\mathrm{RTS}}$	Request-to-Send. This output can be programmed to be negated or asserted automatically by the receiver or the transmitter. When connected to a transmitter's $\overline{\mathrm{U}n\mathrm{CTS}}$, $\overline{\mathrm{U}n\mathrm{RTS}}$ can control serial data flow.

Figure 34-2 shows a signal configuration for a UART/RS-232 interface.

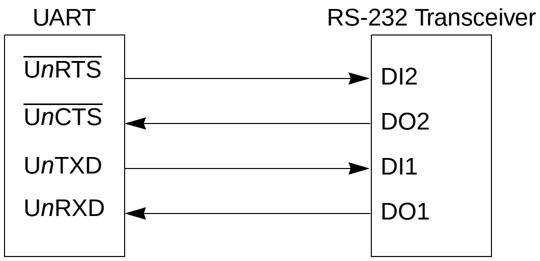

UART
RS-232 Transceiver
$\overline{\mathrm{U}n\mathrm{RTS}}$
$\overline{\mathrm{U}n\mathrm{CTS}}$
U*n*TXD
U*n*RXD
DI2
DO2
DI1
DO1

Figure 34-2. UART/RS-232 Interface

34.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in Section 34.5, "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 34-2. UART Module Memory Map

Address	Register	Width (bit)	Access	Reset Value	Section/Page
UART0 **UART1** **UART2**					
0xFC06_0000 0xFC06_4000 0xFC06_8000	UART Mode Registers[1] (UMR1*n*), (UMR2*n*)	8	R/W	0x00	34.3.1/34-5 34.3.2/34-6
0xFC06_0004 0xFC06_4004 0xFC06_8004	UART Status Register (USR*n*)	8	R	0x00	34.3.3/34-8
	UART Clock Select Register[1](UCSR*n*)	8	W	See Section	34.3.4/34-9
0xFC06_0008 0xFC06_4008 0xFC06_8008	UART Command Registers (UCR*n*)	8	W	0x00	34.3.5/34-9
0xFC06_000C 0xFC06_400C 0xFC06_800C	UART Receive Buffers (URB*n*)	8	R	0xFF	34.3.6/34-11
	UART Transmit Buffers (UTB*n*)	8	W	0x00	34.3.7/34-12
0xFC06_0010 0xFC06_4010 0xFC06_8010	UART Input Port Change Register (UIPCR*n*)	8	R	See Section	34.3.8/34-12
	UART Auxiliary Control Register (UACR*n*)	8	W	0x00	34.3.9/34-13
0xFC06_0014 0xFC06_4014 0xFC06_8014	UART Interrupt Status Register (UISR*n*)	8	R	0x00	34.3.10/34-13
	UART Interrupt Mask Register (UIMR*n*)	8	W	0x00	
0xFC06_0018 0xFC06_4018 0xFC06_8018	UART Baud Rate Generator Register (UBG1*n*)	8	W[2]	0x00	34.3.11/34-15
0xFC06_001C 0xFC06_401C 0xFC06_801C	UART Baud Rate Generator Register (UBG2*n*)	8	W[2]	0x00	34.3.11/34-15
0xFC06_0034 0xFC06_4034 0xFC06_8034	UART Input Port Register (UIP*n*)	8	R	0xFF	34.3.12/34-15
0xFC06_0038 0xFC06_4038 0xFC06_8038	UART Output Port Bit Set Command Register (UOP1*n*)	8	W[2]	0x00	34.3.13/34-16
0xFC06_003C 0xFC06_403C 0xFC06_803C	UART Output Port Bit Reset Command Register (UOP0*n*)	8	W[2]	0x00	34.3.13/34-16

[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.

34.3.1 UART Mode Registers 1 (UMR1*n*)

The UMR1*n* registers control UART module configuration. UMR1*n* can be read or written when the mode register pointer points to it, at RESET or after a RESET MODE REGISTER POINTER command using UCR*n*[MISC]. After UMR1*n* is read or written, the pointer points to UMR2*n*.

Address: 0xFC06_0000 (UMR10)
0xFC06_4000 (UMR11)
0xFC06_8000 (UMR12)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	RXRTS	RXIRQ/ FFULL	ERR	PM		PT	B/C	
Reset:	0	0	0	0	0	0	0	0

[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 34-3. UART Mode Registers 1 (UMR1*n*)

Table 34-3. UMR1*n* Field Descriptions

Field	Description
7 RXRTS	Receiver request-to-send. Allows the $\overline{\mathrm{U}n\mathrm{RTS}}$ output to control the $\overline{\mathrm{U}n\mathrm{CTS}}$ input of the transmitting device to prevent receiver overrun. If the receiver and transmitter are incorrectly programmed for $\overline{\mathrm{U}n\mathrm{RTS}}$ control, $\overline{\mathrm{U}n\mathrm{RTS}}$ control is disabled for both. Transmitter RTS control is configured in UMR2*n*[TXRTS]. 0 The receiver has no effect on $\overline{\mathrm{U}n\mathrm{RTS}}$. 1 When a valid start bit is received, $\overline{\mathrm{U}n\mathrm{RTS}}$ is negated if the UART's FIFO is full. $\overline{\mathrm{U}n\mathrm{RTS}}$ is reasserted when the FIFO has an empty position available.
6 RXIRQ/ FFULL	Receiver interrupt select. 0 RXRDY is the source generating interrupt or DMA requests. 1 FFULL is the source generating interrupt or DMA requests.
5 ERR	Error mode. Configures the FIFO status bits, USR*n*[RB,FE,PE]. 0 Character mode. The USR*n* values reflect the status of the character at the top of the FIFO. ERR must be 0 for correct A/D flag information when in multidrop mode. 1 Block mode. The USR*n* values are the logical OR of the status for all characters reaching the top of the FIFO since the last RESET ERROR STATUS command for the UART was issued. See Section 34.3.5, "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 34-3. UMR1*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>2
PT</td><td>Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11).
<table>
<tr><th>PM</th><th>Parity Mode</th><th>Parity Type (PT= 0)</th><th>Parity Type (PT= 1)</th></tr>
<tr><td>00</td><td>With parity</td><td>Even parity</td><td>Odd parity</td></tr>
<tr><td>01</td><td>Force parity</td><td>Low parity</td><td>High parity</td></tr>
<tr><td>10</td><td>No parity</td><td colspan="2">N/A</td></tr>
<tr><td>11</td><td>Multidrop mode</td><td>Data character</td><td>Address character</td></tr>
</table>
</td></tr>
<tr><td>1–0
B/C</td><td>Bits per character. Selects the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits.
00 5 bits
01 6 bits
10 7 bits
11 8 bits</td></tr>
</table>

34.3.2 UART Mode Register 2 (UMR2*n*)

The UMR2*n* registers control UART module configuration. UMR2*n* can be read or written when the mode register pointer points to it, which occurs after any access to UMR1*n*. UMR2*n* accesses do not update the pointer.

Address: 0xFC06_0000 (UMR20)
0xFC06_4000 (UMR21)
0xFC06_8000 (UMR22)

Access: User read/write[1]

	7	6	5	4	3	2	1	0
R W	CM		TXRTS	TXCTS	SB			
Reset:	0	0	0	0	0	0	0	0

[1] After UMR1*n* is read or written, the pointer points to UMR2*n*

Figure 34-4. UART Mode Registers 2 (UMR2*n*)

Table 34-4. UMR2*n* Field Descriptions

Field	Description
7–6 CM	Channel mode. Selects a channel mode. Section 34.4.3, “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 $\overline{\mathrm{U}n\mathrm{RTS}}$ to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same UART for $\overline{\mathrm{U}n\mathrm{RTS}}$ control is not permitted and disables $\overline{\mathrm{U}n\mathrm{RTS}}$ control for both. 0 The transmitter has no effect on $\overline{\mathrm{U}n\mathrm{RTS}}$. 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 $\overline{\mathrm{U}n\mathrm{CTS}}$ has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of $\overline{\mathrm{U}n\mathrm{CTS}}$ each time it is ready to send a character. If $\overline{\mathrm{U}n\mathrm{CTS}}$ 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 $\overline{\mathrm{U}n\mathrm{CTS}}$ is asserted. Changes in $\overline{\mathrm{U}n\mathrm{CTS}}$ 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.

SB	5 Bits	6–8 Bits
0000	1.063	0.563
0001	1.125	0.625
0010	1.188	0.688
0011	1.250	0.750
0100	1.313	0.813
0101	1.375	0.875
0110	1.438	0.938
0111	1.500	1.000

SB	5–8 Bits
1000	1.563
1001	1.625
1010	1.688
1011	1.750
1100	1.813
1101	1.875
1110	1.938
1111	2.000

34.3.3 UART Status Registers (USR*n*)

The USR*n* registers show the status of the transmitter, the receiver, and the FIFO.

Address: 0xFC06_0004 (USR0)
0xFC06_4004 (USR1)
0xFC06_8004 (USR2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	RB	FE	PE	OE	TXEMP	TXRDY	FFULL	RXRDY
W								
Reset:	0	0	0	0	0	0	0	0

Figure 34-5. UART Status Registers (USR*n*)

Table 34-5. 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 34-5. 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.

34.3.4 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 34.4.1, “Transmitter/Receiver Clock Source.” The transmitter and receiver can use different clock sources. To use the internal bus clock for both, set UCSR*n* to 0xDD.

Address: 0xFC06_0004 (UCSR0)
0xFC06_4004 (UCSR1)
0xFC06_8004 (UCSR2)

Access: User write-only

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

Note: The RCS and TCS reset values are set so the receiver and transmiter use the prescaled internal bus clock as their clock source.

Figure 34-6. UART Clock Select Registers (UCSR*n*)

Table 34-6. 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

34.3.5 UART Command Registers (UCR*n*)

The UCRs supply commands to the UART. Only multiple commands that do not conflict can be specified in a single write to a UCR*n*. For example, RESET TRANSMITTER and ENABLE TRANSMITTER cannot be specified in one command.

Address: 0xFC06_0008 (UCR0)
0xFC06_4008 (UCR1)
0xFC06_8008 (UCR2)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	0	MISC			TC		RC	
Reset:	0	0	0	0	0	0	0	0

Figure 34-7. UART Command Registers (UCR*n*)

Table 34-7 describes UCR*n* fields and commands. Examples in Section 34.4.2, “Transmitter and Receiver Operating Modes,” show how these commands are used.

Table 34-7. UCR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved, must be cleared.</td></tr>
<tr><td>6–4
MISC</td><td>MISC Field (this field selects a single command)
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>000</td><td>NO COMMAND</td><td>—</td></tr>
<tr><td>001</td><td>RESET MODE REGISTER POINTER</td><td>Causes the mode register pointer to point to UMR1<i>n</i>.</td></tr>
<tr><td>010</td><td>RESET RECEIVER</td><td>Immediately disables the receiver, clears USR<i>n</i>[FFULL,RXRDY], and reinitializes the receiver FIFO pointer. No other registers are altered. Because it places the receiver in a known state, use this command instead of RECEIVER DISABLE when reconfiguring the receiver.</td></tr>
<tr><td>011</td><td>RESET TRANSMITTER</td><td>Immediately disables the transmitter and clears USR<i>n</i>[TXEMP,TXRDY]. No other registers are altered. Because it places the transmitter in a known state, use this command instead of TRANSMITTER DISABLE when reconfiguring the transmitter.</td></tr>
<tr><td>100</td><td>RESET ERROR STATUS</td><td>Clears USR<i>n</i>[RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received.</td></tr>
<tr><td>101</td><td>RESET BREAK – CHANGE INTERRUPT</td><td>Clears the delta break bit, UISR<i>n</i>[DB].</td></tr>
<tr><td>110</td><td>START BREAK</td><td>Forces U<i>n</i>TXD low. If the transmitter is empty, break may be delayed up to one bit time. If the transmitter is active, break starts when character transmission completes. Break is delayed until any character in the transmitter shift register is sent. Any character in the transmitter holding register is sent after the break. Transmitter must be enabled for the command to be accepted. This command ignores the state of $\overline{\text{U}n\text{CTS}}$.</td></tr>
<tr><td>111</td><td>STOP BREAK</td><td>Causes U<i>n</i>TXD to go high (mark) within two bit times. Any characters in the transmit buffer are sent.</td></tr>
</table>
</td></tr>
</table>

Table 34-7. UCR*n* Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–2
TC</td><td>Transmit command field. Selects a single transmit command.
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>00</td><td>NO ACTION TAKEN</td><td>Causes the transmitter to stay in its current mode: if the transmitter is enabled, it remains enabled; if the transmitter is disabled, it remains disabled.</td></tr>
<tr><td>01</td><td>TRANSMITTER ENABLE</td><td>Enables operation of the UART’s transmitter. USR<i>n</i>[TXEMP,TXRDY] are set. If the transmitter is already enabled, this command has no effect.</td></tr>
<tr><td>10</td><td>TRANSMITTER DISABLE</td><td>Terminates transmitter operation and clears USR<i>n</i>[TXEMP,TXRDY]. If a character is being sent when the transmitter is disabled, transmission completes before the transmitter becomes inactive. If the transmitter is already disabled, the command has no effect.</td></tr>
<tr><td>11</td><td>—</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
<tr><td>1–0
RC</td><td>Receive command field. Selects a single receive command.
<table>
<tr><th></th><th>Command</th><th>Description</th></tr>
<tr><td>00</td><td>NO ACTION TAKEN</td><td>Causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.</td></tr>
<tr><td>01</td><td>RECEIVER ENABLE</td><td>If the UART module is not in multidrop mode (UMR1<i>n</i>[PM] ≠ 11), RECEIVER ENABLE enables the UART's receiver and forces it into search-for-start-bit state. If the receiver is already enabled, this command has no effect.</td></tr>
<tr><td>10</td><td>RECEIVER DISABLE</td><td>Disables the receiver immediately. Any character being received is lost. The command does not affect receiver status bits or other control registers. If the UART module is programmed for local loopback or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, the command has no effect.</td></tr>
<tr><td>11</td><td>—</td><td>Reserved, do not use.</td></tr>
</table></td></tr>
</table>

34.3.6 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 34-18). RB contains the character in the receiver.

Address: 0xFC06_000C (URB0)
0xFC06_400C (URB1)
0xFC06_800C (URB2)

Access: User read-only

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

Figure 34-8. UART Receive Buffer (URB*n*)

34.3.7 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 34-9 shows UTB*n*. TB contains the character in the transmit buffer.

Address: 0xFC06_000C (UTB0)
0xFC06_400C (UTB1)
0xFC06_800C (UTB2)

Access: User write-only

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

Figure 34-9. UART Transmit Buffer (UTB*n*)

34.3.8 UART Input Port Change Registers (UIPCR*n*)

The UIPCRs hold the current state and the change-of-state for $\overline{\text{U}n\text{CTS}}$.

Address: 0xFC06_0010 (UIPCR0)
0xFC06_4010 (UIPCR1)
0xFC06_8010 (UIPCR2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	0	0	0	COS	1	1	1	CTS
W								
Reset:	0	0	0	0	1	1	1	$\overline{\text{U}n\text{CTS}}$

Figure 34-10. UART Input Port Changed Registers (UIPCR*n*)

Table 34-8. UIPCR*n* Field Descriptions

Field	Description
7–5	Reserved
4 COS	Change of state (high-to-low or low-to-high transition). 0 No change-of-state since the CPU last read UIPCR*n*. Reading UIPCR*n* clears UISR*n*[COS]. 1 A change-of-state longer than 25–50 µs occurred on the $\overline{\mathrm{U}n\mathrm{CTS}}$ input. UACR*n* can be programmed to generate an interrupt to the CPU when a change of state is detected.
3–1	Reserved
0 CTS	Current state of clear-to-send. Starting two serial clock periods after reset, CTS reflects the state of $\overline{\mathrm{U}n\mathrm{CTS}}$. If $\overline{\mathrm{U}n\mathrm{CTS}}$ is detected asserted at that time, COS is set, which initiates an interrupt if UACR*n*[IEC] is enabled. 0 The current state of the $\overline{\mathrm{U}n\mathrm{CTS}}$ input is asserted. 1 The current state of the $\overline{\mathrm{U}n\mathrm{CTS}}$ input is deasserted.

34.3.9 UART Auxiliary Control Register (UACR*n*)

The UACRs control the input enable.

Address: 0xFC06_0010 (UACR0)
0xFC06_4010 (UACR1)
0xFC06_8010 (UACR2)

Access: User write-only

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

Figure 34-11. UART Auxiliary Control Registers (UACR*n*)

Table 34-9. UACR*n* Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 IEC	Input enable control. 0 Setting the corresponding UIPCR*n* bit has no effect on UISR*n*[COS]. 1 UISR*n*[COS] is set and an interrupt is generated when the UIPCR*n*[COS] is set by an external transition on the $\overline{\mathrm{U}n\mathrm{CTS}}$ input (if UIMR*n*[COS] = 1).

34.3.10 UART Interrupt Status/Mask Registers (UISR*n*/UIMR*n*)

The UISRs provide status for all potential interrupt sources. UISR*n* contents are masked by UIMR*n*. If corresponding UISR*n* and UIMR*n* bits are set, internal interrupt output is asserted. If a UIMR*n* bit is cleared, state of the corresponding UISR*n* bit has no effect on the output.

The UISR*n* and UIMR*n* registers share the same space in memory. Reading this register provides the user with interrupt status, while writing controls the mask bits.

NOTE

True status is provided in the UISR*n* regardless of UIMR*n* settings. UISR*n* is cleared when the UART module is reset.

Address: 0xFC06_0014 (UISR0)
0xFC06_4014 (UISR1)
0xFC06_8014 (UISR2)

Access: User read/write

	7	6	5	4	3	2	1	0
R (UISR*n*)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
W (UIMR*n*)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
Reset:	0	0	0	0	0	0	0	0

Figure 34-12. UART Interrupt Status/Mask Registers (UISR*n*/UIMR*n*)

Table 34-10. UISR*n*/UIMR*n* Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>7
COS</td><td>Change-of-state.
0 UIPCR<i>n</i>[COS] is not selected.
1 Change-of-state occurred on $\overline{\text{U}n\text{CTS}}$ and was programmed in UACR<i>n</i>[IEC] to cause an interrupt.</td></tr>
<tr><td>6–3</td><td>Reserved, must be cleared.</td></tr>
<tr><td>2
DB</td><td>Delta break.
0 No new break-change condition to report. Section 34.3.5, “UART Command Registers (UCRn),” describes the RESET BREAK-CHANGE INTERRUPT command.
1 The receiver detected the beginning or end of a received break.</td></tr>
<tr><td>1
FFULL/
RXRDY</td><td>Status of FIFO or receiver, depending on UMR1[FFULL/RXRDY] bit. Duplicate of USR<i>n</i>[FIFO] and USR<i>n</i>[RXRDY]
<table>
<tr><th rowspan="2">UIMR<i>n</i>
[FFULL/RXRDY]</th><th rowspan="2">UISR<i>n</i>
[FFULL/RXRDY]</th><th colspan="2">UMR1<i>n</i>[FFULL/RXRDY]</th></tr>
<tr><th>0 (RXRDY)</th><th>1 (FIFO)</th></tr>
<tr><td>0</td><td>0</td><td>Receiver not ready</td><td>FIFO not full</td></tr>
<tr><td>1</td><td>0</td><td>Receiver not ready</td><td>FIFO not full</td></tr>
<tr><td>0</td><td>1</td><td>Receiver is ready,
Do not interrupt</td><td>FIFO is full,
Do not interrupt</td></tr>
<tr><td>1</td><td>1</td><td>Receiver is ready,
interrupt</td><td>FIFO is full,
interrupt</td></tr>
</table></td></tr>
<tr><td>0
TXRDY</td><td>Transmitter ready. This bit is the duplication of USR<i>n</i>[TXRDY].
0 The transmitter holding register was loaded by the CPU or the transmitter is disabled. Characters loaded into the transmitter holding register when TXRDY is cleared are not sent.
1 The transmitter holding register is empty and ready to be loaded with a character.</td></tr>
</table>

34.3.11 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 34.4.1.2.1, “Internal Bus Clock Baud Rates.”

Address: 0xFC06_0018 (UBG10)
0xFC06_4018 (UBG11)
0xFC06_8018 (UBG12)

Access: User write-only

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

Figure 34-13. UART Baud Rate Generator Registers (UBG1*n*)

Address: 0xFC06_001C (UBG20)
0xFC06_401C (UBG21)
0xFC06_801C (UBG22)

Access: User write-only

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

Figure 34-14. 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.

34.3.12 UART Input Port Register (UIP*n*)

The UIP*n* registers show the current state of the $\overline{\mathrm{U}n\mathrm{CTS}}$ input.

Address: 0xFC06_0034 (UIP0)
0xFC06_4034 (UIP1)
0xFC06_8034 (UIP2)

Access: User read-only

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

Figure 34-15. UART Input Port Registers (UIP*n*)

Table 34-11. UIP*n* Field Descriptions

Field	Description
7–1	Reserved
0 CTS	Current state of clear-to-send. The $\overline{\text{U}n\text{CTS}}$ value is latched and reflects the state of the input pin when UIP*n* is read. **Note:** This bit has the same function and value as UIPCR*n*[CTS]. 0 The current state of the $\overline{\text{U}n\text{CTS}}$ input is logic 0. 1 The current state of the $\overline{\text{U}n\text{CTS}}$ input is logic 1.

34.3.13 UART Output Port Command Registers (UOP1*n*/UOP0*n*)

The $\overline{\text{U}n\text{RTS}}$ output can be asserted by writing a 1 to UOP1*n*[RTS] and negated by writing a 1 to UOP0*n*[RTS].

Address: 0xFC06_0038 (UOP10)
0xFC06_003C (UOP00)
0xFC06_4038 (UOP11)
0xFC06_403C (UOP01)
0xFC06_8038 (UOP12)
0xFC06_803C (UOP02)

Access: User write-only

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

Figure 34-16. UART Output Port Command Registers (UOP1*n*/UOP0*n*)

Table 34-12. UOP1*n*/UOP0*n* Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 RTS	Output port output. Controls assertion (UOP1)/negation (UOP0) of $\overline{\text{U}n\text{RTS}}$ output. 0 Not affected. 1 Asserts $\overline{\text{U}n\text{RTS}}$ in UOP1. Negates $\overline{\text{U}n\text{RTS}}$ in UOP0.

34.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

34.4.1 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.

34.4.1.1 Programmable Divider

As Figure 34-17 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 34.3.11, "UART Baud Rate Generator Registers (UBG1n/UBG2n)."

The choice of DTIN or internal bus clock is programmed in the UCSR.

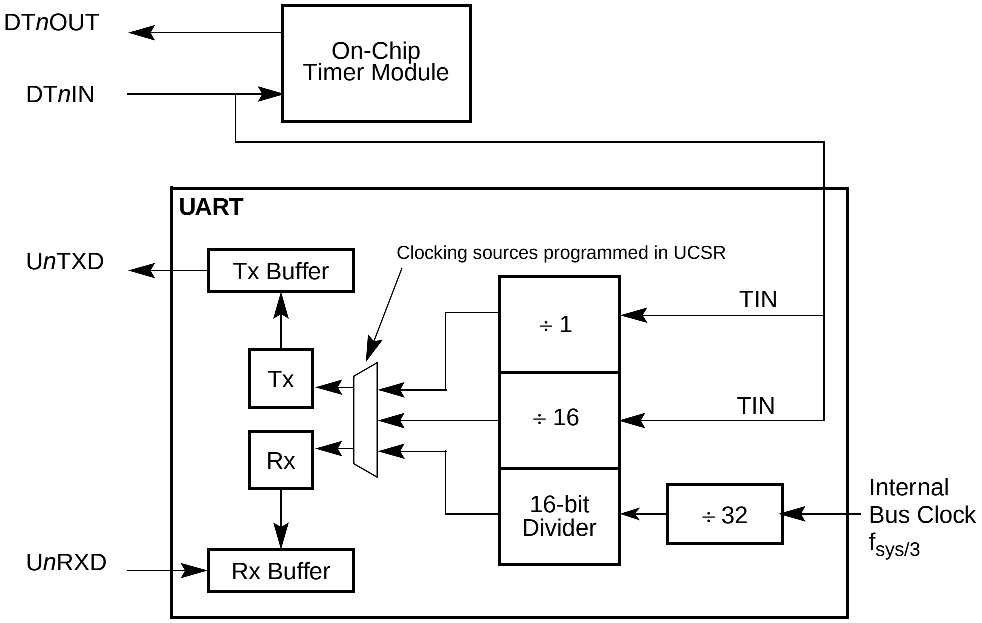


Figure 34-17. Clocking Source Diagram

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.

34.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

34.4.1.2.1 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:

$$\text{Baudrate} = \frac{f_{sys/3}}{[32 \times \text{divider}]} \qquad \textit{Eqn. 34-1}$$

Using a 80-MHz internal bus clock and letting baud rate equal 9600, then

$$\text{Divider} = \frac{80\text{MHz}}{[32 \times 9600]} = 260(\text{decimal}) = 0\text{x}0104(\text{hexadecimal}) \qquad \textit{Eqn. 34-2}$$

Therefore, UBG1*n* equals 0x01 and UBG2*n* equals 0x04.

34.4.1.2.2 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:

$$\text{Baudrate} = \frac{f_{extc}}{(16\text{ or }1)} \qquad \textit{Eqn. 34-3}$$

34.4.2 Transmitter and Receiver Operating Modes

Figure 34-18 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 34.3, “Memory Map/Register Definition.”

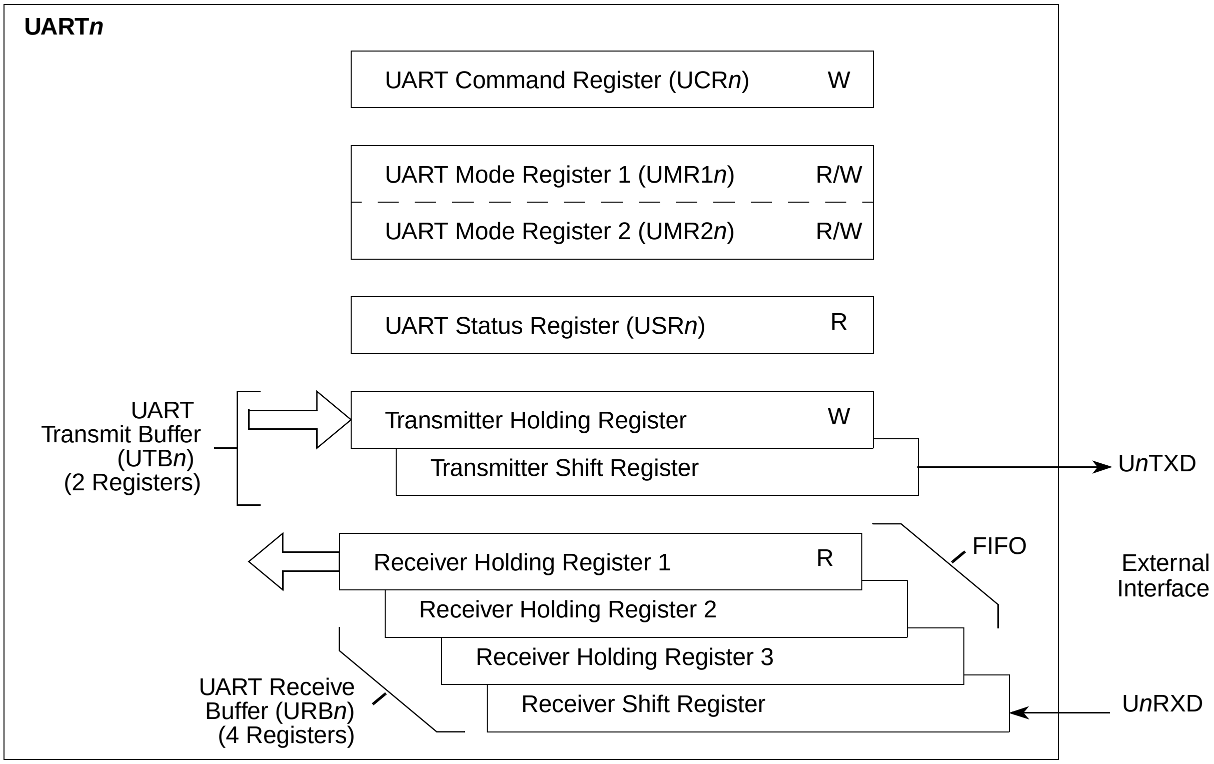

UARTn
UART Command Register (UCRn) W
UART Mode Register 1 (UMR1n) R/W
UART Mode Register 2 (UMR2n) R/W
UART Status Register (USRn) R
UART Transmit Buffer (UTBn) (2 Registers)
Transmitter Holding Register W
Transmitter Shift Register
UnTXD
FIFO
External Interface
Receiver Holding Register 1 R
Receiver Holding Register 2
Receiver Holding Register 3
Receiver Shift Register
UART Receive Buffer (URBn) (4 Registers)
UnRXD

Figure 34-18. Transmitter and Receiver Functional Diagram

34.4.2.1 Transmitter

The transmitter is enabled through the UART command register (UCR*n*). When it is ready to accept a character, UART sets USR*n*[TXRDY]. The transmitter converts parallel data from the CPU to a serial bit stream on U*n*TXD. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The lsb is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

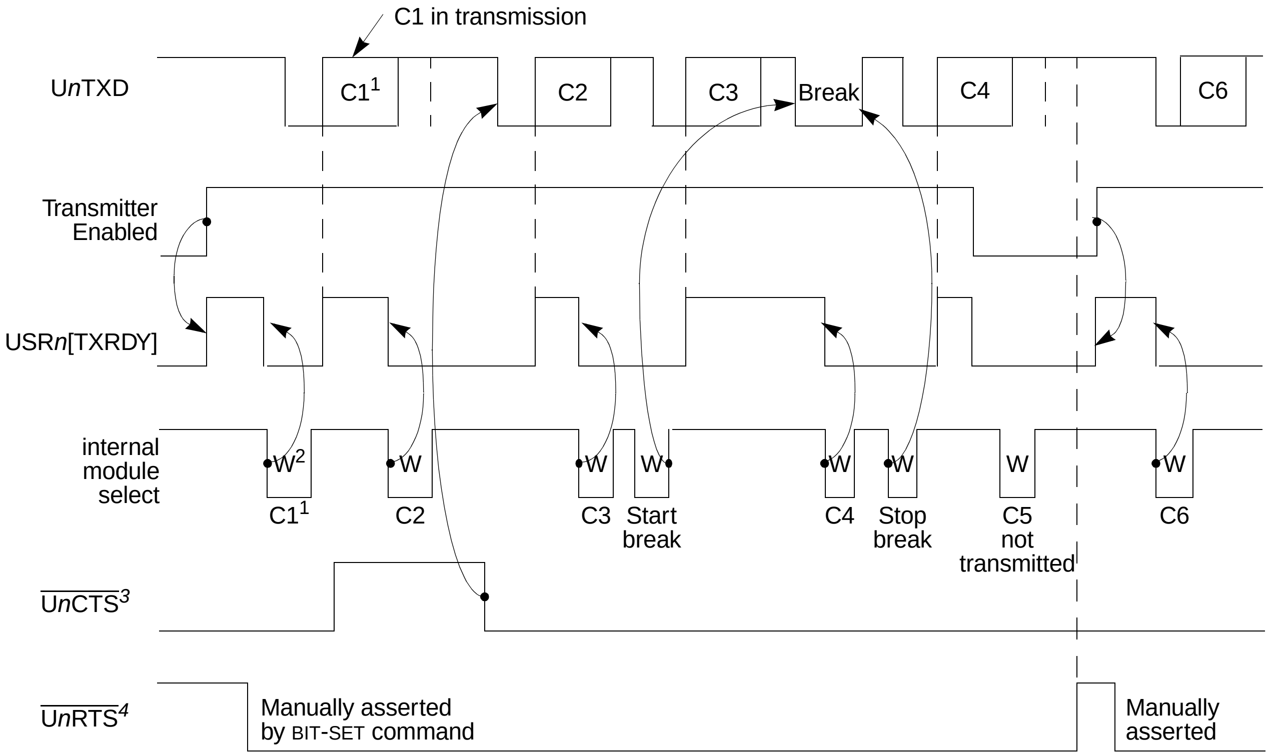
After the stop bits are sent, if no new character is in the transmitter holding register, the U*n*TXD output remains high (mark condition) and the transmitter empty bit (USR*n*[TXEMP]) is set. Transmission resumes and TXEMP is cleared when the CPU loads a new character into the UART transmit buffer (UTB*n*). If the transmitter receives a disable command, it continues until any character in the transmitter shift register is completely sent.

If the transmitter is reset through a software command, operation stops immediately (see Section 34.3.5, "UART Command Registers (UCRn)"). The transmitter is reenabled through the UCR*n* to resume operation after a disable or software reset.

If the clear-to-send operation is enabled, $\overline{\mathrm{U}n\mathrm{CTS}}$ must be asserted for the character to be transmitted. If $\overline{\mathrm{U}n\mathrm{CTS}}$ is negated in the middle of a transmission, the character in the shift register is sent and U*n*TXD remains in mark state until $\overline{\mathrm{U}n\mathrm{CTS}}$ is reasserted. If transmitter is forced to send a continuous low condition by issuing a SEND BREAK command, transmitter ignores the state of $\overline{\mathrm{U}n\mathrm{CTS}}$.

If the transmitter is programmed to automatically negate $\overline{\mathrm{U}n\mathrm{RTS}}$ when a message transmission completes, $\overline{\mathrm{U}n\mathrm{RTS}}$ must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and $\overline{\mathrm{U}n\mathrm{RTS}}$ is appropriately programmed, $\overline{\mathrm{U}n\mathrm{RTS}}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting $\overline{\mathrm{U}n\mathrm{RTS}}$ before the next message is sent.

Figure 34-19 shows the functional timing information for the transmitter.



[1] Cn = transmit characters
[2] W = write
[3] UMR2n[TXCTS] = 1
[4] UMR2n[TXRTS] = 1

Figure 34-19. Transmitter Timing Diagram

34.4.2.2 Receiver

The receiver is enabled through its UCR*n*, as described in Section 34.3.5, "UART Command Registers (UCRn)."

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on U*n*RXD, the state of U*n*RXD is sampled eight times on the edge of the bit time clock starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If U*n*RXD is sampled high, start bit is invalid and the search for the valid start bit begins again.

If U*n*RXD remains low, a valid start bit is assumed. The receiver continues sampling the input at one-bit time intervals at the theoretical center of the bit until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the U*n*RXD input is sampled on the rising edge of the programmed clock source. The lsb is received first. The data then transfers to a receiver holding register and USR*n*[RXRDY] is set. If the character is less than 8 bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, receiver immediately looks for the next start bit. However, if a non-zero character is received without a stop bit (framing error) and U*n*RXD remains low for one-half of the bit period after the stop bit is sampled, receiver operates as if a new start bit were detected. Parity error, framing error, overrun error, and received break conditions set the respective PE, FE, OE, and RB error and break flags in the USR*n* at the received character boundary. They are valid only if USR*n*[RXRDY] is set.

If a break condition is detected (U*n*RXD is low for the entire character including the stop bit), a character of all 0s loads into the receiver holding register and USR*n*[RB,RXRDY] are set. U*n*RXD must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. The receiver places the damaged character in the Rx FIFO and sets the corresponding USR*n* error bits and USR*n*[RXRDY]. Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets USR*n*[RB,RXRDY].

Figure 34-20 shows receiver functional timing.

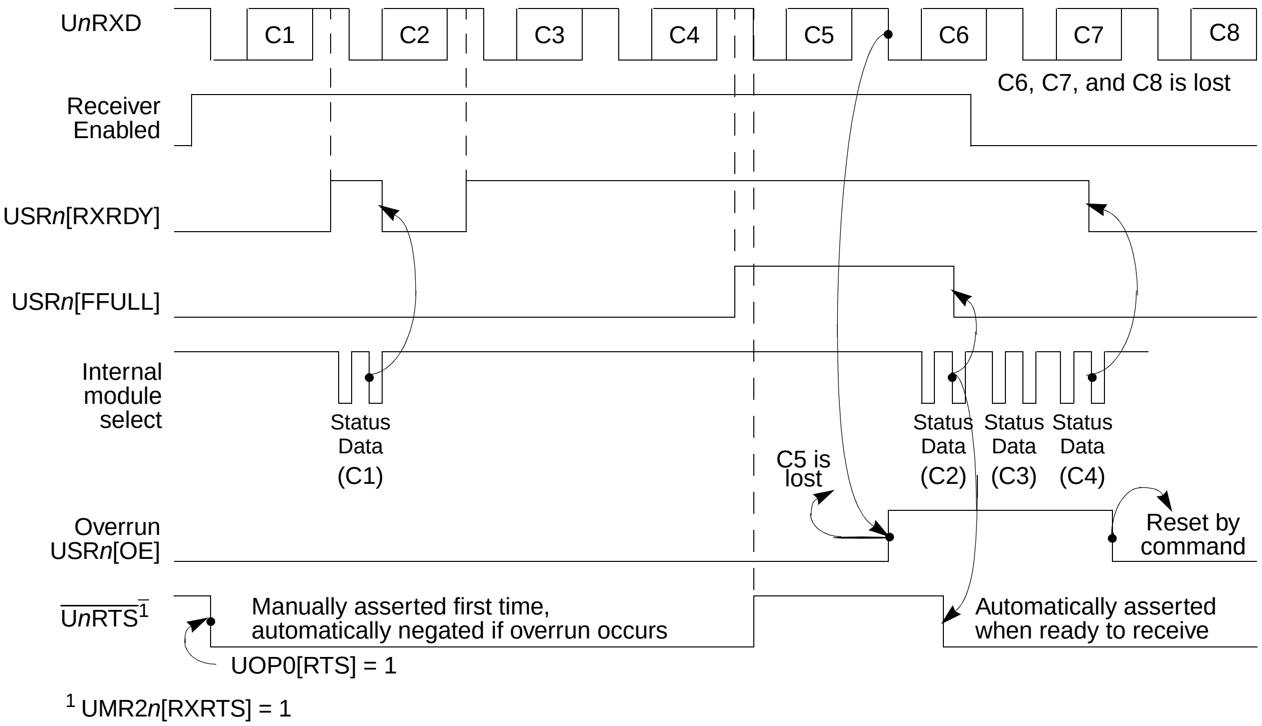


Figure 34-20. Receiver Timing Diagram

34.4.2.3 FIFO

The FIFO is used in the UART's receive buffer logic. The FIFO consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the U*n*RXD (see Figure 34-18). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Therefore, data flowing from the receiver to the CPU is quadruple-buffered.

In addition to the data byte, three status bits—parity error (PE), framing error (FE), and received break (RB)—are appended to each data character in the FIFO; overrun error (OE) is not appended. By programming the ERR bit in the UART's mode register (UMR1*n*), status is provided in character or block modes.

USR*n*[RXRDY] is set when at least one character is available to be read by the CPU. A read of the receive buffer produces an output of data from the top of the FIFO. After the read cycle, the data at the top of the FIFO and its associated status bits are popped and the receiver shift register can add new data at the bottom of the FIFO. The FIFO-full status bit (FFULL) is set if all three positions are filled with data. The RXRDY or FFULL bit can be selected to cause an interrupt and TXRDY or RXRDY can be used to generate a DMA request.

The two error modes are selected by UMR1*n*[ERR]:

- In character mode (UMR1*n*[ERR] = 0), status is given in the USR*n* for the character at the top of the FIFO.
- In block mode, the USR*n* shows a logical OR of all characters reaching the top of the FIFO since the last RESET ERROR STATUS command. Status is updated as characters reach the top of the FIFO. Block mode offers a data-reception speed advantage where the software overhead of error-checking each character cannot be tolerated. However, errors are not detected until the check is performed at the end of an entire message—the faulting character is not identified.

In either mode, reading the USR*n* does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The USR*n* should be read before reading the receive buffer. If all three receiver holding registers are full, a new character is held in the receiver shift register until space is available. However, if a second new character is received, the contents of the character in the receiver shift register is lost, the FIFOs are unaffected, and USR*n*[OE] is set when the receiver detects the start bit of the new overrunning character.

To support flow control, the receiver can be programmed to automatically negate and assert $\overline{\mathrm{U}n\mathrm{RTS}}$, in which case the receiver automatically negates $\overline{\mathrm{U}n\mathrm{RTS}}$ when a valid start bit is detected and the FIFO is full. The receiver asserts $\overline{\mathrm{U}n\mathrm{RTS}}$ when a FIFO position becomes available; therefore, connecting $\overline{\mathrm{U}n\mathrm{RTS}}$ to the $\overline{\mathrm{U}n\mathrm{CTS}}$ 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, $\overline{\mathrm{U}n\mathrm{RTS}}$ control, all receiver status bits, interrupts, and DMA requests are reset. No more characters are received until the receiver is reenabled.

34.4.3 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 34.3, "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.

34.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in Figure 34-21, 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.

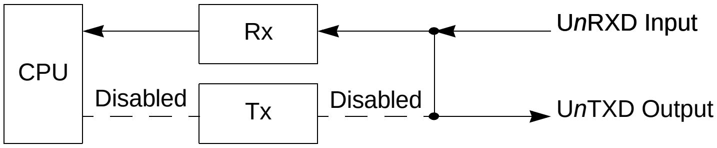


Figure 34-21. Automatic Echo

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.

34.4.3.2 Local Loopback Mode

Figure 34-22 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.

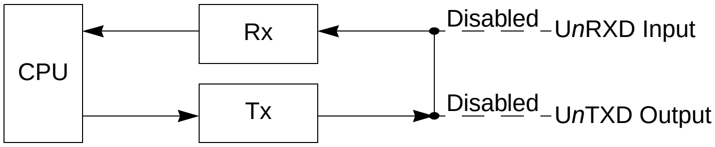


Figure 34-22. Local Loopback

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.

34.4.3.3 Remote Loopback Mode

In remote loopback mode, shown in Figure 34-23, 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.

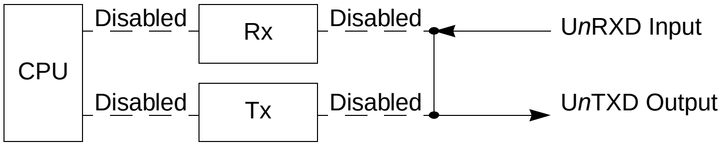


Figure 34-23. Remote Loopback

34.4.4 Multidrop Mode

Setting UMR1*n*[PM] programs the UART to operate in a wake-up mode for multidrop or multiprocessor applications. In this mode, a master can transmit an address character followed by a block of data characters targeted for one of up to 256 slave stations.

Although slave stations have their receivers disabled, they continuously monitor the master's data stream. When the master sends an address character, the slave receiver notifies its respective CPU by setting USR*n*[RXRDY] and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Unaddressed slave stations continue monitoring the data stream. Data fields in the data stream are separated by an address character. After a slave receives a block of data, its CPU disables the receiver and repeats the process. Functional timing information for multidrop mode is shown in Figure 34-24.

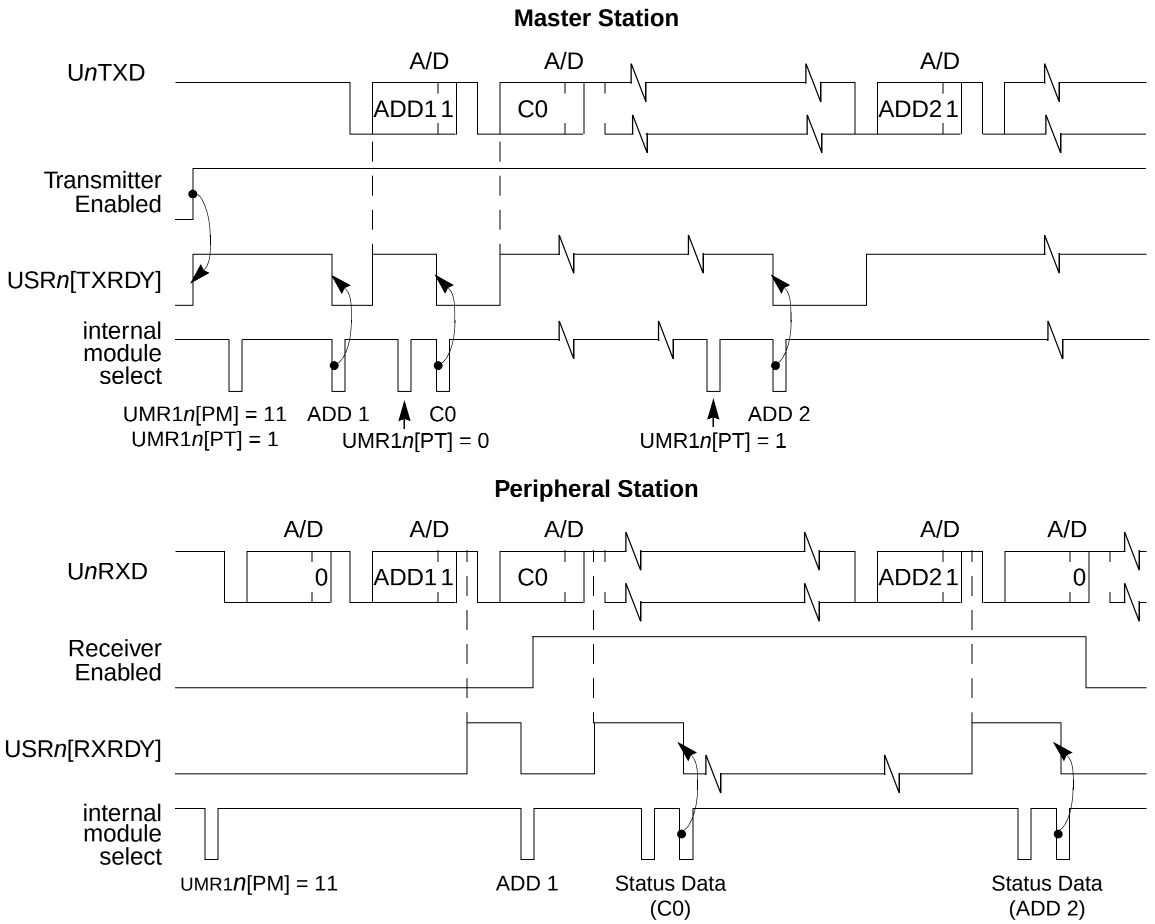


Figure 34-24. Multidrop Mode Timing Diagram

A character sent from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. A/D equals 1 indicates an address character; A/D equals 0 indicates a data character. The polarity of A/D is selected through UMR1*n*[PT]. UMR1*n* should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RXRDY bit and loads the character into the receiver holding register FIFO provided the received A/D bit is a 1 (address tag). The character is discarded if the received A/D bit is 0 (data tag). If the receiver is enabled, all received characters are transferred to the CPU through the receiver holding register during read operations.

In either case, data bits load into the data portion of the FIFO while the A/D bit loads into the status portion of the FIFO normally used for a parity error (USR*n*[PE]).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may continues containing error detection and correction information. If 8-bit characters are not required, one way to provide error detection is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

34.4.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

34.4.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

34.4.5.2 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.

34.5 Initialization/Application Information

The software flowchart, Figure 34-25, consists of:

- UART module initialization—These routines consist of SINIT and CHCHK (See Sheet 1 p. 34-29 and Sheet 2 p. 34-30). 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. 34-32 and Sheet 5 p. 34-33) 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. 34-32), 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.

34.5.1 Interrupt and DMA Request Initialization

34.5.1.1 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 15.2.9.1, “Interrupt Sources,” for details on interrupt assignments for the UART modules.

1. Initialize the appropriate ICR*x* register in the interrupt controller.
2. Unmask appropriate bits in IMR in the interrupt controller.

3. Unmask appropriate bits in the core’s status register (SR) to enable interrupts.
4. If TXRDY or RXRDY generates interrupt requests, verify that the corresponding UART DMA channels are not enabled.
5. Initialize interrupts in the UART, see Table 34-13.

Table 34-13. 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

34.5.1.2 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 34-14 shows the DMA requests.

Table 34-14. UART DMA Requests

Register	Bit	DMA Request
UISR*n*	1	Receive DMA request
UISR*n*	0	Transmit DMA request

34.5.2 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.

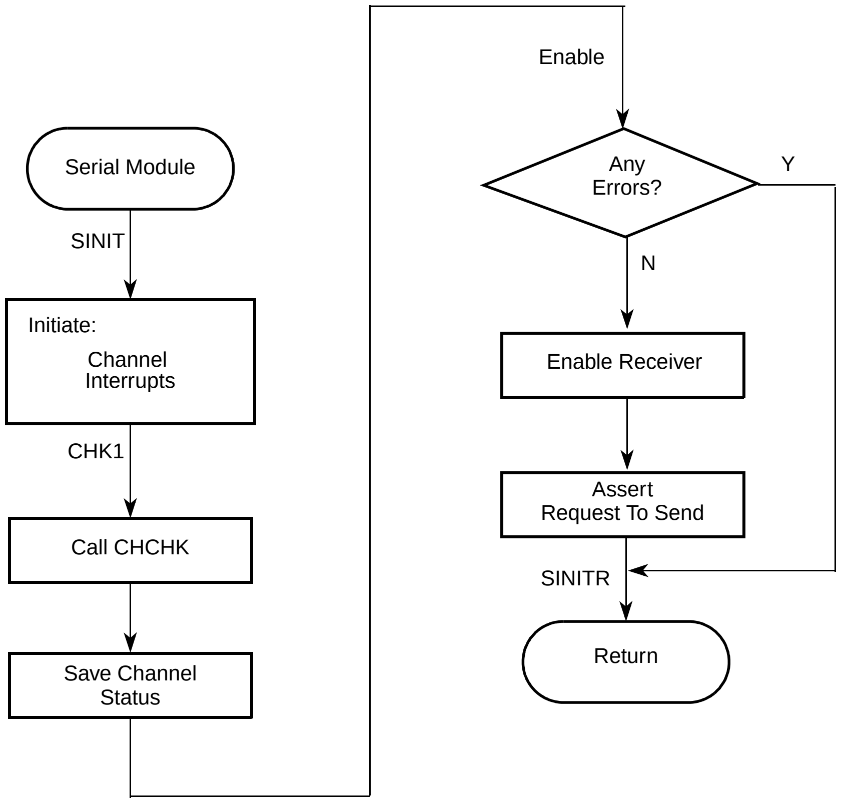


Figure 34-25. UART Mode Programming Flowchart (Sheet 1 of 5)

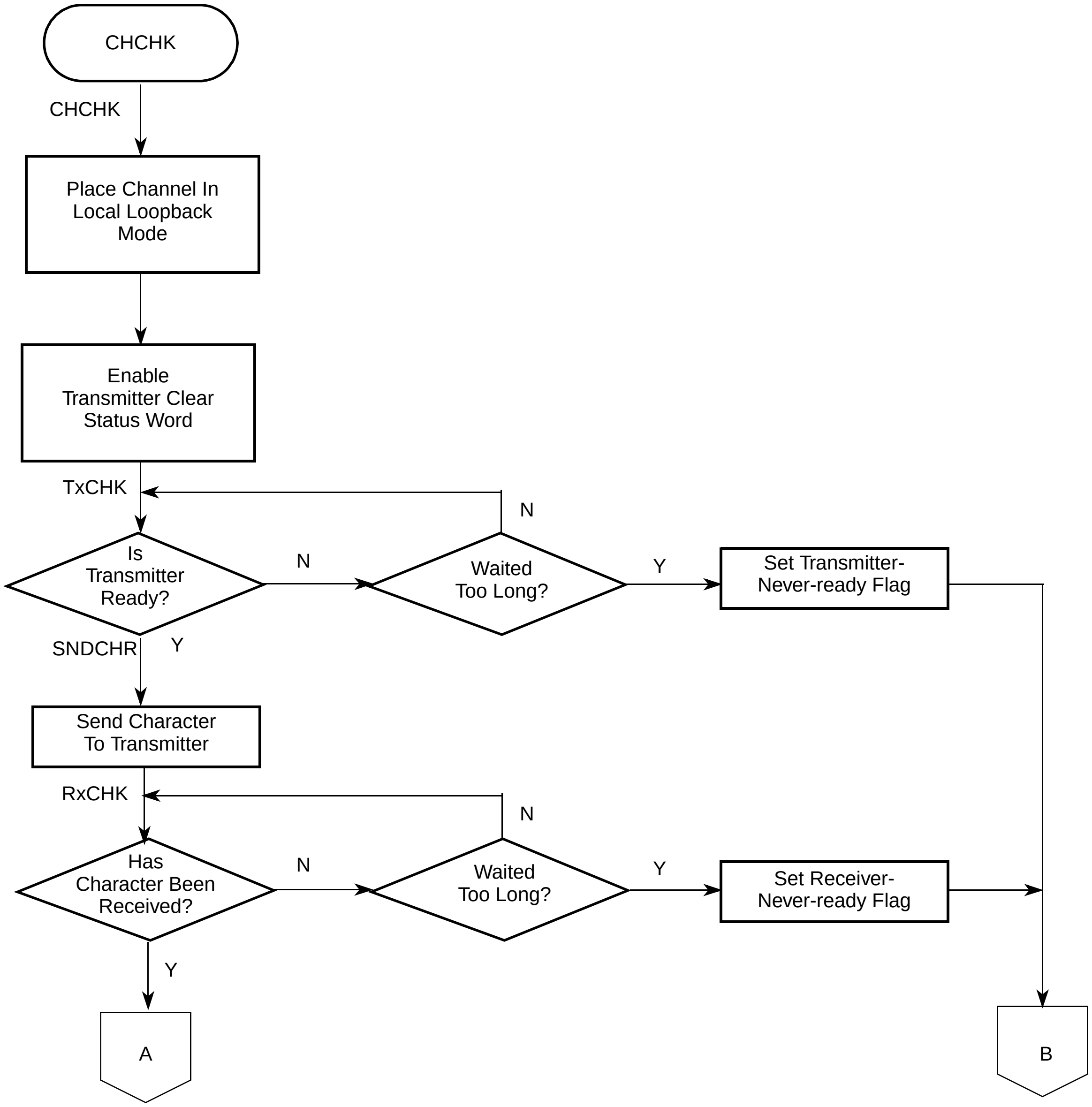

CHCHK
CHCHK
Place Channel In Local Loopback Mode
Enable Transmitter Clear Status Word
TxCHK
N
Is Transmitter Ready?
N
Waited Too Long?
Y
Set Transmitter-Never-ready Flag
SNDCHR
Y
Send Character To Transmitter
RxCHK
N
Has Character Been Received?
N
Waited Too Long?
Y
Set Receiver-Never-ready Flag
Y
A
B

Figure 34-25. UART Mode Programming Flowchart (Sheet 2 of 5)

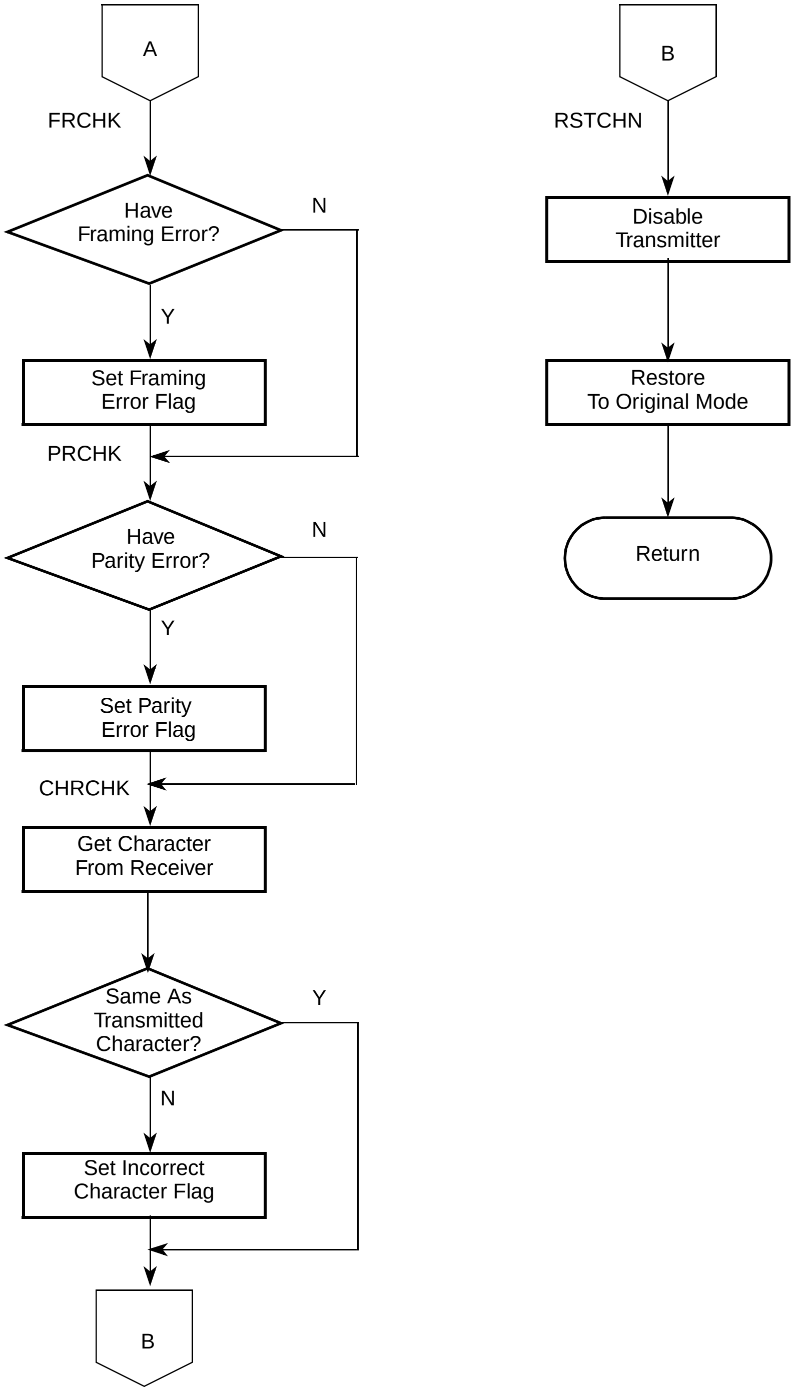


Figure 34-25. UART Mode Programming Flowchart (Sheet 3 of 5)

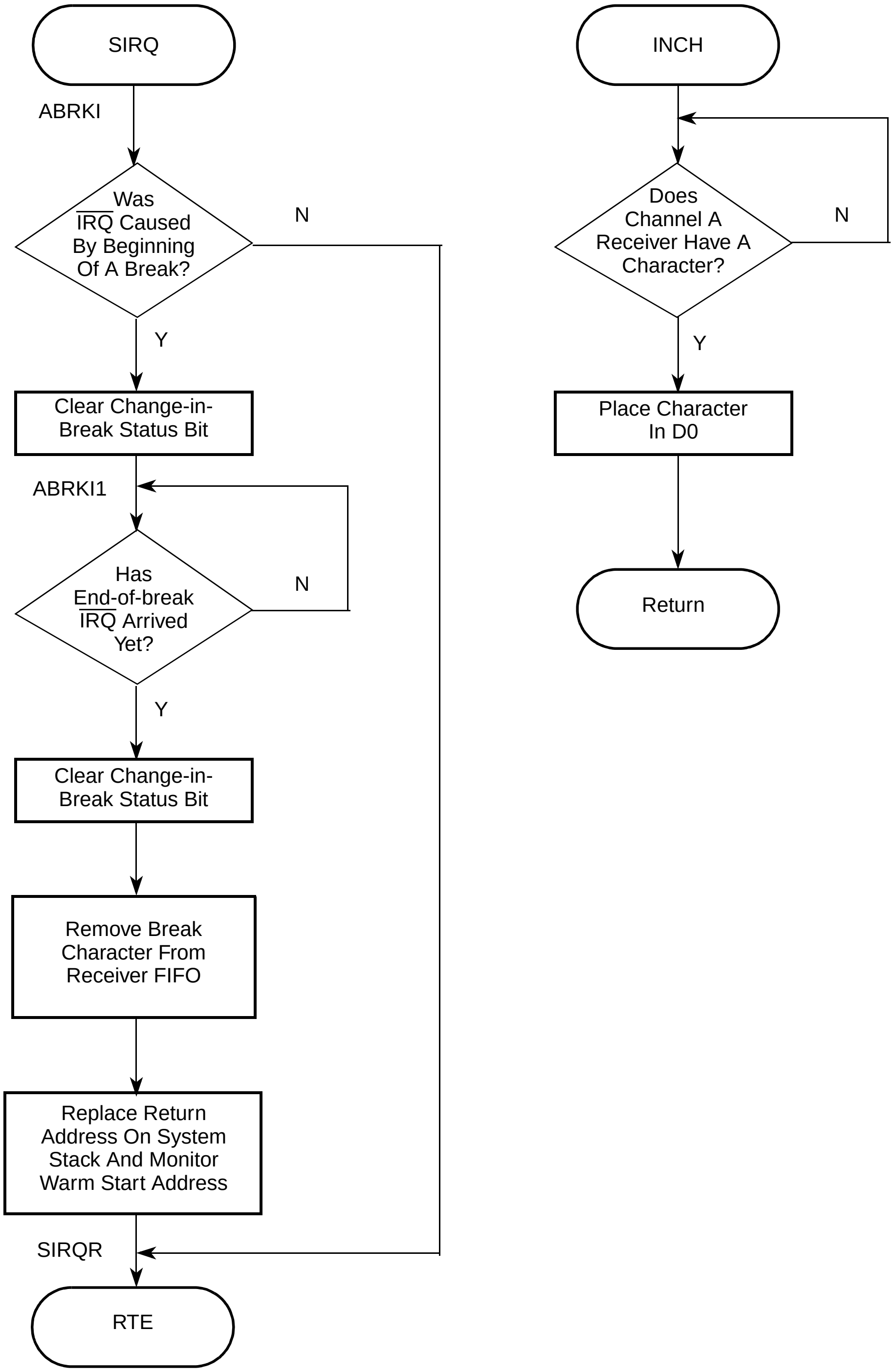

SIRQ
ABRKI
Was IRQ Caused By Beginning Of A Break?
N
Y
Clear Change-in-Break Status Bit
ABRKI1
Has End-of-break IRQ Arrived Yet?
N
Y
Clear Change-in-Break Status Bit
Remove Break Character From Receiver FIFO
Replace Return Address On System Stack And Monitor Warm Start Address
SIRQR
RTE
INCH
Does Channel A Receiver Have A Character?
N
Y
Place Character In D0
Return

Figure 34-25. UART Mode Programming Flowchart (Sheet 4 of 5)

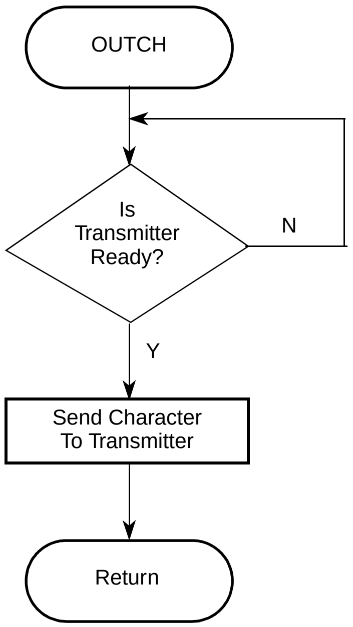


Figure 34-25. UART Mode Programming Flowchart (Sheet 5 of 5)

Chapter 35
I²C Interface

35.1 Introduction

This chapter describes the I²C module, clock synchronization, and I²C programming model registers. It also provides extensive programming examples.

35.1.1 Block Diagram

Figure 35-1 is a I²C module block diagram, illustrating the interaction of the registers described in Section 35.2, “Memory Map/Register Definition”.

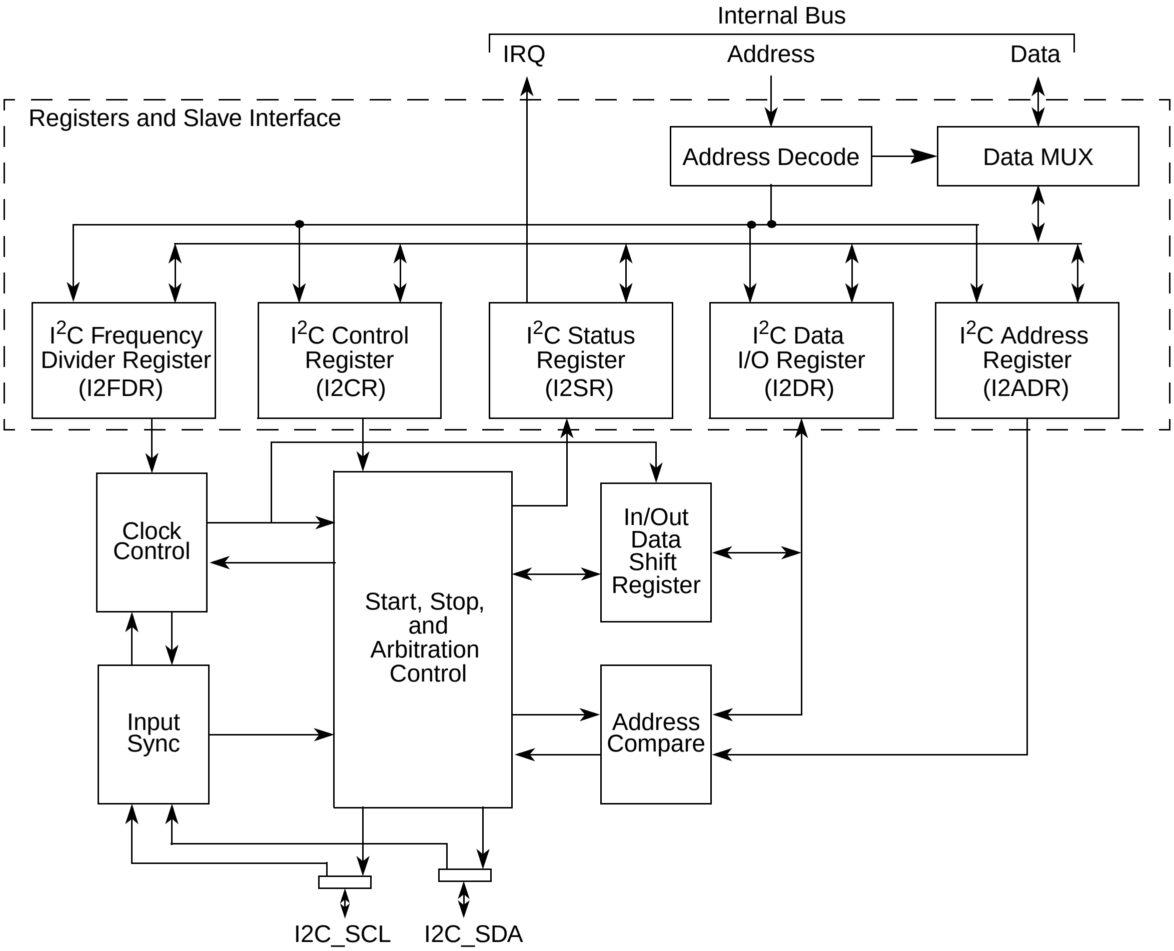

Internal Bus
IRQ
Address
Data
Registers and Slave Interface
Address Decode
Data MUX
I²C Frequency Divider Register (I2FDR)
I²C Control Register (I2CR)
I²C Status Register (I2SR)
I²C Data I/O Register (I2DR)
I²C Address Register (I2ADR)
Clock Control
Start, Stop, and Arbitration Control
In/Out Data Shift Register
Input Sync
Address Compare
I2C_SCL
I2C_SDA

Figure 35-1. I²C Module Block Diagram

35.1.2 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 14, “Pin-Multiplexing and Control”) prior to configuring the I^{2}C module.

35.1.3 Features

The I^{2}C module has these key features:

- Compatibility with I^{2}C bus standard version 2.1
- Multiple-master operation
- Software-programmable for one of 50 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

35.2 Memory Map/Register Definition

The below table lists the configuration registers used in the I^{2}C interface.

Table 35-1. I^{2}C Module Memory Map

Address	Register	Access	Reset Value	Section/Page
0xFC05_8000	I^{2}C Address Register (I2ADR)	R/W	0x00	35.2.1/35-3
0xFC05_8004	I^{2}C Frequency Divider Register (I2FDR)	R/W	0x00	35.2.2/35-3
0xFC05_8008	I^{2}C Control Register (I2CR)	R/W	0x00	35.2.3/35-4
0xFC05_800C	I^{2}C Status Register (I2SR)	R/W	0x81	35.2.4/35-5
0xFC05_8010	I^{2}C Data I/O Register (I2DR)	R/W	0x00	35.2.5/35-6

35.2.1 I^{2}C Address Register (I2ADR)

I2ADR holds the address the I^{2}C responds to when addressed as a slave. It is not the address sent on the bus during the address transfer when the module is performing a master transfer.

Address: 0xFC05_8000 (I2ADR)   Access: User read/write

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

Figure 35-2. I^{2}C Address Register (I2ADR)

Table 35-2. I2ADR Field Descriptions

Field	Description
7–1 ADR	Slave address. Contains the specific slave address to be used by the I^{2}C module. Slave mode is the default I^{2}C mode for an address match on the bus.
0	Reserved, must be cleared.

35.2.2 I^{2}C Frequency Divider Register (I2FDR)

The I2FDR, shown in Figure 35-3, provides a programmable prescaler to configure the I^{2}C clock for bit-rate selection.

Address: 0xFC05_8004 (I2FDR)   Access: User read/write

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

Figure 35-3. I^{2}C Frequency Divider Register (I2FDR)

Table 35-3. I2FDR Field Descriptions

Field	Description
7–6	Reserved, must be cleared.
5–0 IC	I^2C 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

35.2.3 I^2C Control Register (I2CR)

I2CR enables the I^2C module and the I^2C 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 35-4. I^2C Control Register (I2CR)

Table 35-4. I2CR Field Descriptions

Field	Description
7 IEN	I^{2}C enable. Controls the software reset of the entire I^{2}C module. If the module is enabled in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next START condition is detected. Master mode is not aware that the bus is busy; initiating a start cycle may corrupt the current bus cycle, ultimately causing the current master or the I^{2}C module to lose arbitration, after which bus operation returns to normal. 0 The I^{2}C module is disabled, but registers can be accessed. 1 The I^{2}C module is enabled. This bit must be set before any other I2CR bits have any effect.
6 IIEN	I^{2}C interrupt enable. 0 I^{2}C module interrupts are disabled, but currently pending interrupt condition is not cleared. 1 I^{2}C module interrupts are enabled. An I^{2}C interrupt occurs if I2SR[IIF] is also set.
5 MSTA	Master/slave mode select bit. If the master loses arbitration, MSTA is cleared without generating a STOP signal. 0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode. 1 Master mode. Changing MSTA from 0 to 1 signals a START on the bus and selects master mode.
4 MTX	Transmit/receive mode select bit. Selects the direction of master and slave transfers. 0 Receive 1 Transmit. When the device is addressed as a slave, software must set MTX according to I2SR[SRW]. In master mode, MTX must be set according to the type of transfer required. Therefore, when the MCU addresses a slave device, MTX is always 1.
3 TXAK	Transmit acknowledge enable. Specifies the value driven onto I2C_SDA during acknowledge cycles for master and slave receivers. Writing TXAK applies only when the I^{2}C bus is a receiver. 0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (acknowledge bit = 1).
2 RSTA	Repeat start. Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration. 0 No repeat start 1 Generates a repeated START condition.
1–0	Reserved, must be cleared.

35.2.4 I^{2}C Status Register (I2SR)

I2SR contains bits that indicate transaction direction and status.

Address: 0xFC05_800C (I2SR) Access: User read/write

	7	6	5	4	3	2	1	0
R	ICF	IAAS	IBB	IAL	0	SRW	IIF	RXAK
W								
Reset:	1	0	0	0	0	0	0	1

Figure 35-5. I^{2}C Status Register (I2SR)

Table 35-5. 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.

35.2.5 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 35-6. I²C Data I/O Register (I2DR)

Table 35-6. 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.

35.3 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 35.4.1, “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.

35.3.1 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 35-7). 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.

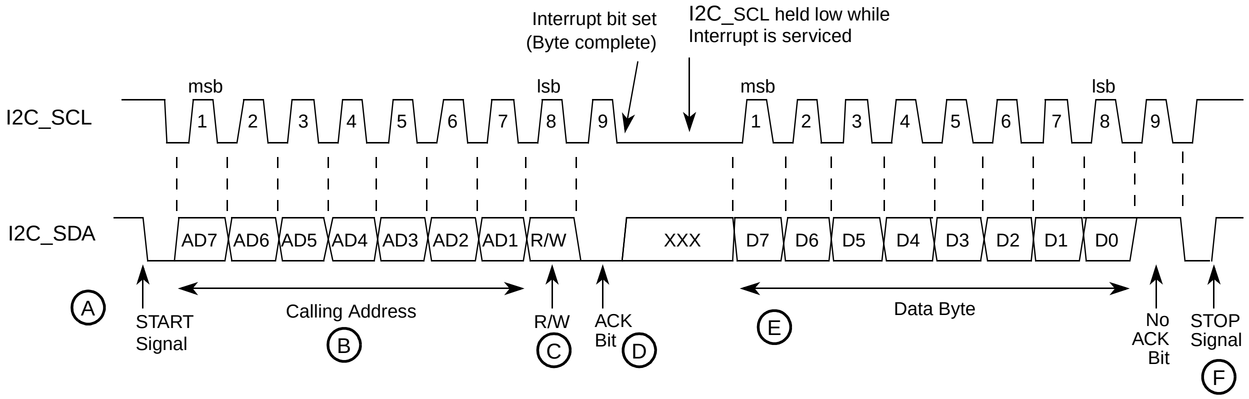


Figure 35-7. I^{2}C Standard Communication Protocol

35.3.2 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 I^{2}C 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.

35.3.3 Data Transfer

When successful slave addressing is achieved, data transfer can proceed (see E in Figure 35-7) 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 35-7 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 35-8.

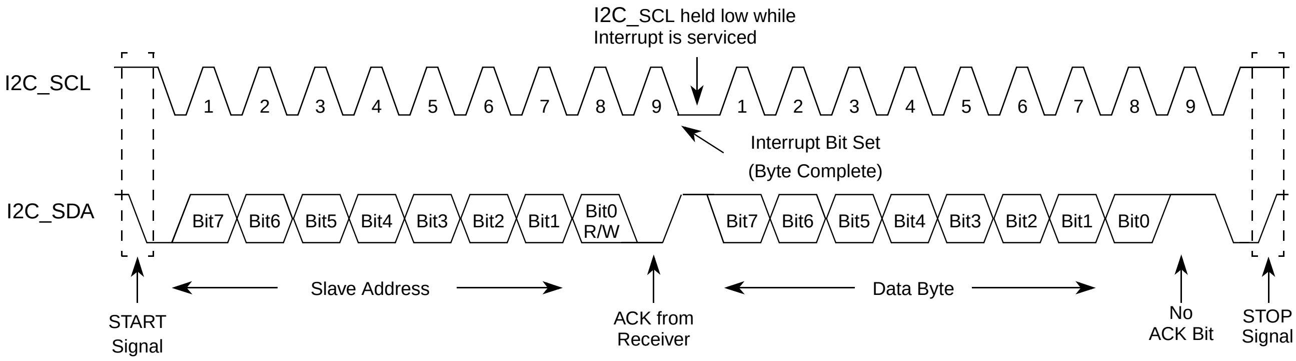


Figure 35-8. Data Transfer

35.3.4 Acknowledge

The transmitter releases the I2C_SDA line high during the acknowledge clock pulse as shown in Figure 35-9. 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 35-10 and discussed in Section 35.3.6, “Repeated START”) to start a new calling sequence.

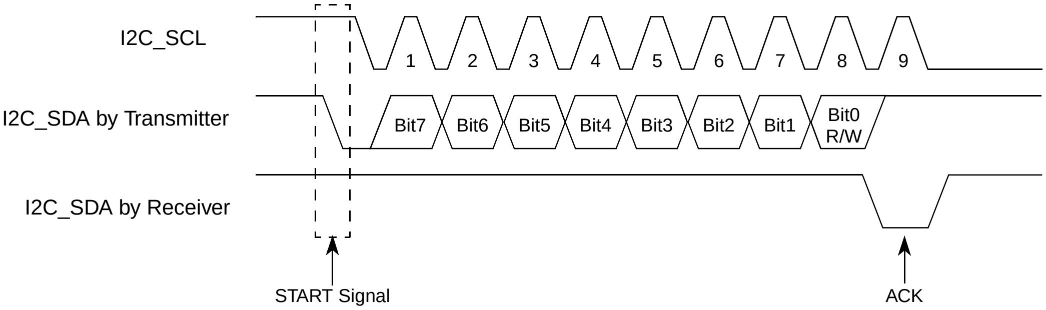


Figure 35-9. Acknowledgement by Receiver

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 35-9).

35.3.5 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 35-7). 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 35.3.6, “Repeated START.”

35.3.6 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 35-10. 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.

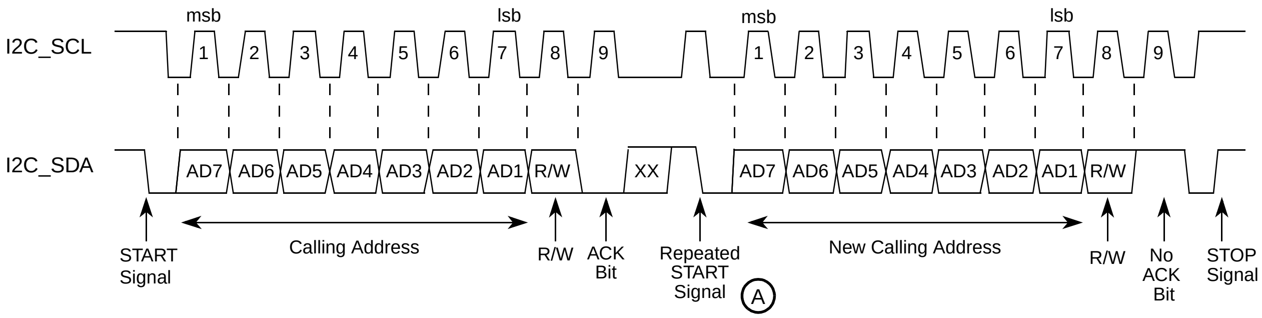


Figure 35-10. Repeated START

Various combinations of read/write formats are then possible:

- The first example in Figure 35-11 is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.
- The second example in Figure 35-11 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 35-11, 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.

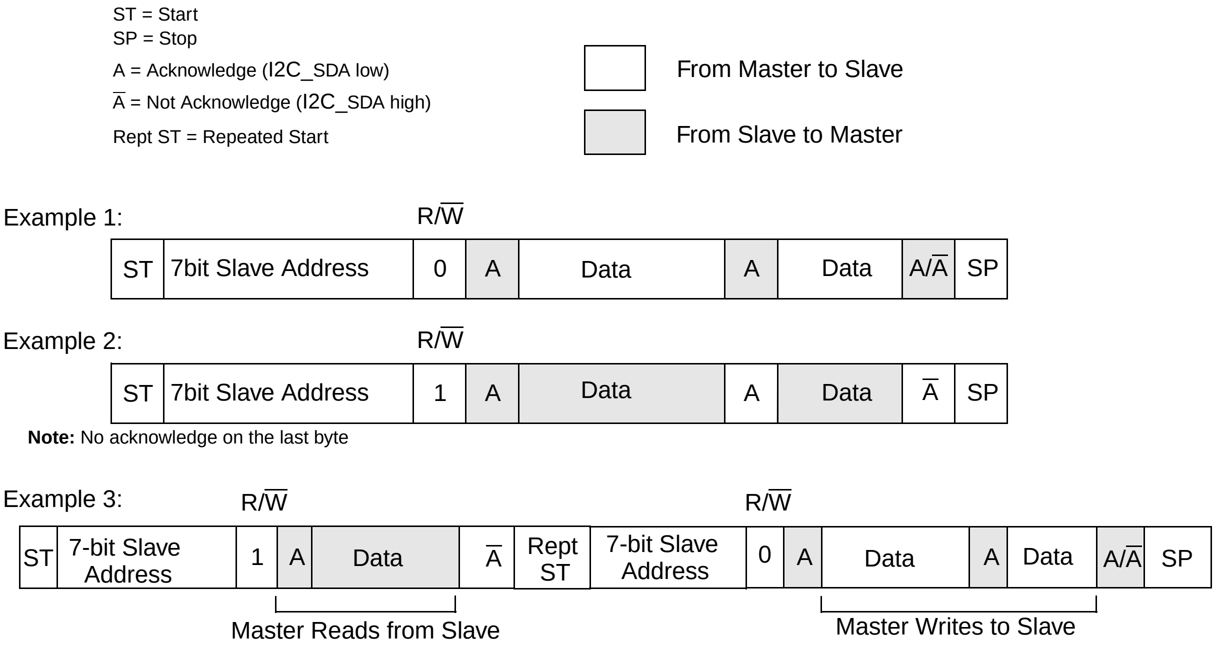

ST = Start
SP = Stop
A = Acknowledge (I2C_SDA low)
$\overline{\text{A}}$ = Not Acknowledge (I2C_SDA high)
Rept ST = Repeated Start
From Master to Slave
From Slave to Master
Example 1:
R/$\overline{\text{W}}$
ST | 7bit Slave Address | 0 | A | Data | A | Data | A/$\overline{\text{A}}$ | SP
Example 2:
R/$\overline{\text{W}}$
ST | 7bit Slave Address | 1 | A | Data | A | Data | $\overline{\text{A}}$ | SP
Note: No acknowledge on the last byte
Example 3:
R/$\overline{\text{W}}$
R/$\overline{\text{W}}$
ST | 7-bit Slave Address | 1 | A | Data | $\overline{\text{A}}$ | Rept ST | 7-bit Slave Address | 0 | A | Data | A | Data | A/$\overline{\text{A}}$ | SP
Master Reads from Slave
Master Writes to Slave

Figure 35-11. Data Transfer, Combined Format

35.3.7 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 35-12). 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.

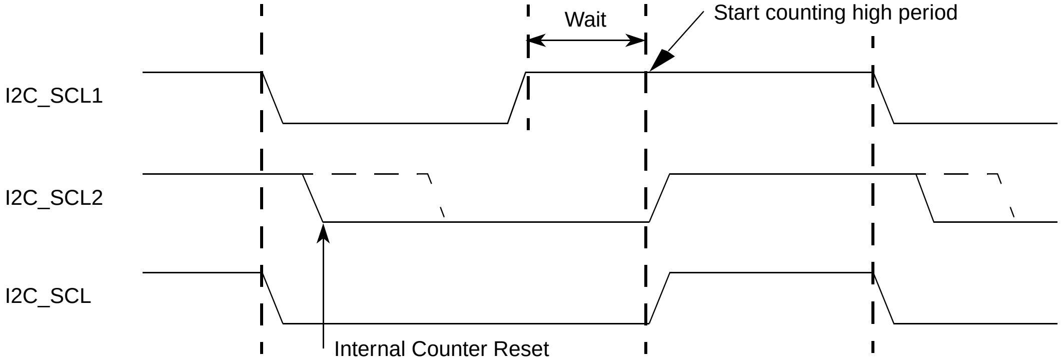


Figure 35-12. Clock Synchronization

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 35-13). 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.

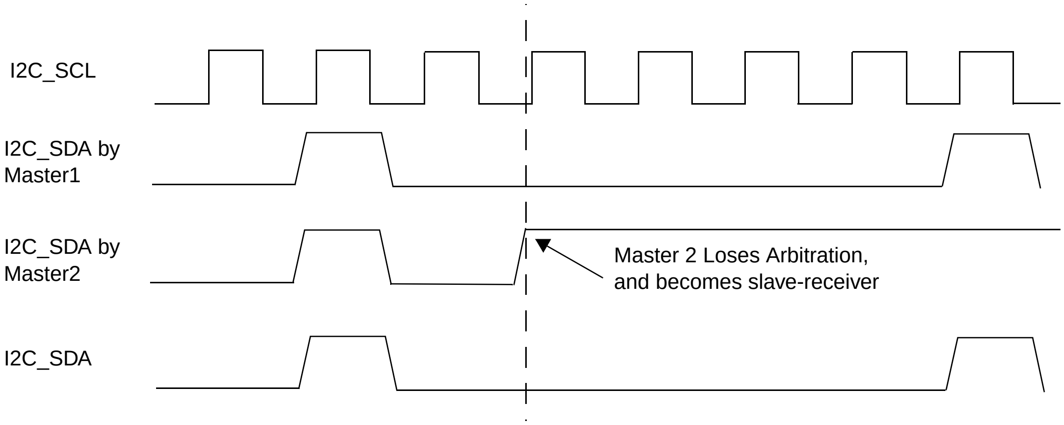


Figure 35-13. Arbitration Procedure

35.3.8 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.

35.4 Initialization/Application Information

The following examples show programming for initialization, signaling START, post-transfer software response, signaling STOP, and generating a repeated START.

35.4.1 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 35.2.2, “I2C Frequency Divider Register (I2FDR).”
2. Update the I2ADR to define its slave address.
3. Set I2CR[IEN] to enable the I^{2}C bus interface system.
4. Modify the I2CR to select or deselect master/slave mode, transmit/receive mode, and interrupt-enable or not.

NOTE

If I2SR[IBB] is set when the I^{2}C bus module is enabled, execute the following pseudocode sequence before proceeding with normal initialization code. This issues a STOP command to the slave device, placing it in idle state as if it were power-cycled on.

```
I2CR = 0x0
I2CR = 0xA0
dummy read of I2DR
I2SR = 0x0
I2CR = 0x0
I2CR = 0x80       ; re-enable
```

35.4.2 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.

35.4.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR[ICF], which indicates one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR[IIEN]. Software must first clear I2SR[IIF] in the interrupt routine. Reading from I2DR in receive mode or writing to I2DR in transmit mode can clear I2SR[ICF].

Software can service the I^{2}C I/O in the main program by monitoring the IIF bit if the interrupt function is disabled. Polling should monitor IIF rather than ICF, because that operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode; the address is sent. If master receive mode is required, I2CR[MTX] should be toggled.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

The following is an example of a software response by a master transmitter in the interrupt routine (see Figure 35-14).

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.

35.4.4 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.

35.4.5 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.

35.4.6 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.

35.4.7 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.

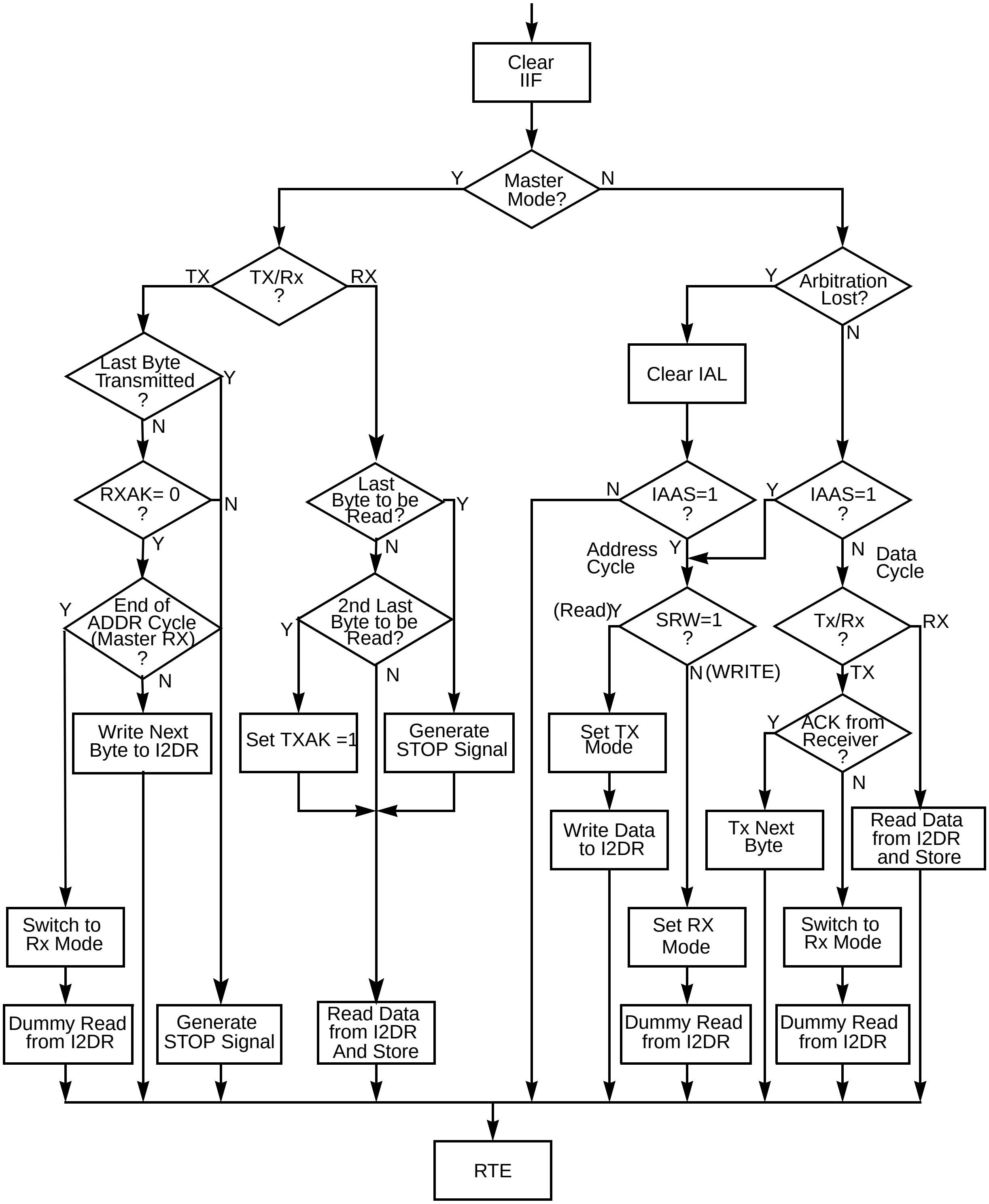


Figure 35-14. Flow-Chart of Typical I^{2}C Interrupt Routine

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.

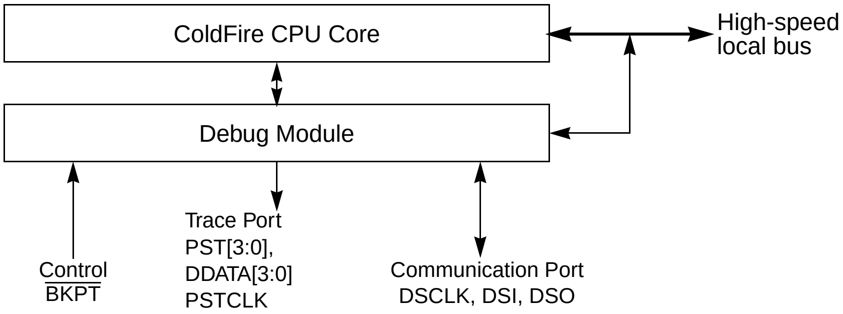


Figure 36-1. Processor/Debug Module Interface

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 additional 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) $\overline{\text{BKPT}}$ configurable interrupt (CSR[BKD]) Level 1 and level 2 triggers on OR condition, in addition to AND SYNC_PC command to display the processor’s current PC
B+	1001	—	3 additional PC breakpoint registers PBR1–3

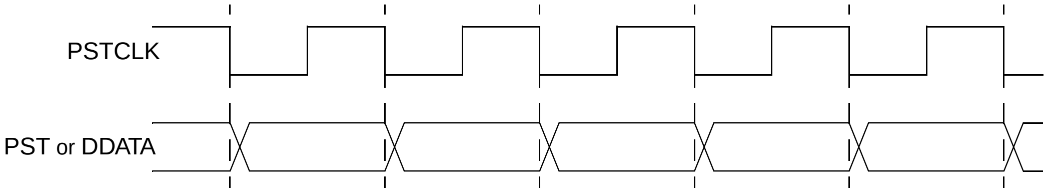
36.2 Signal Descriptions

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 ($\overline{\text{BKPT}}$)	Input requests a manual breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status signals (PST[3:0]) as the value 0xF. If CSR[BKD] is set (disabling normal $\overline{\text{BKPT}}$ functionality), asserting $\overline{\text{BKPT}}$ generates a debug interrupt exception in the processor.

Table 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.

Table 36-3. 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. 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 Else Reserved
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, $\overline{\text{BKPT}}$ 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+ 1111 Revision D+PSTB
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 $\overline{\text{BKPT}}$ input signal functionality, and allows the assertion of this pin to generate a debug interrupt. 0 Normal operation 1 $\overline{\text{BKPT}}$ is edge-sensitive: a high-to-low edge on $\overline{\text{BKPT}}$ signals a debug interrupt to the ColdFire core. The processor makes this interrupt request pending until the next sample point occurs, when the exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for nesting debug interrupts.
17 PCD	PSTCLK disable. 0 PSTCLK is fully operational 1 Disables the generation of the PSTCLK and PSTDDATA output signals, and forces these signals to remain quiescent **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)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>4–3
TT</td><td>Transfer Type. Compared with the local bus transfer type signals.
00 Normal processor access
01 Reserved
10 Emulator mode access
11 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.</td></tr>
<tr><td>2–0
TM</td><td>Transfer Modifier. Compared with the local bus transfer modifier signals, which give supplemental information for each transfer type. These bits also define the TM encoding for BDM memory commands (for backward compatibility).
<table>
<tr><th>TM</th><th>TT=00
(normal mode)</th><th>TT=10
(emulator mode)</th></tr>
<tr><td>000</td><td>Explicit cache line push</td><td>Reserved</td></tr>
<tr><td>001</td><td>User data access</td><td>Reserved</td></tr>
<tr><td>010</td><td>User code access</td><td>Reserved</td></tr>
<tr><td>011</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>100</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>101</td><td>Supervisor data access</td><td>Emulator mode access</td></tr>
<tr><td>110</td><td>Supervisor code access</td><td>Emulator code access</td></tr>
<tr><td>111</td><td>Reserved</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

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

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>31–30
TRC</td><td>Trigger Response Control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on DDATA.
00 Display on DDATA only
01 Processor halt
10 Debug interrupt
11 Reserved</td></tr>
<tr><td>29
L2EBL</td><td>Enable Level 2 Breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 2 breakpoints
1 Enables all level 2 breakpoint triggers</td></tr>
<tr><td>28–22
L2ED</td><td>Enable Level 2 Data Breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>28</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>27</td><td>Lower data word.</td></tr>
<tr><td>26</td><td>Upper data word.</td></tr>
<tr><td>25</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>24</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>23</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>22</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>21
L2DI</td><td>Level 2 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
</table>

Table 36-8. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20–18
L2EA</td><td>Enable Level 2 Address Breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>17
L2EPC</td><td>Enable Level 2 PC Breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint where the trigger is defined by the logical summation of:
$$(\text{PBR0 and } \overline{\text{PBMR}}) \mid \text{PBR1} \mid \text{PBR2} \mid \text{PBR3} \qquad \textit{Eqn. 36-1}$$</td></tr>
<tr><td>16
L2PCI</td><td>Level 2 PC Breakpoint Invert.
0 The PC breakpoint is defined within the region defined by PBR<i>n</i> and PBMR.
1 The PC breakpoint is defined outside the region defined by PBR<i>n</i> and PBMR.</td></tr>
<tr><td>15
L2T</td><td>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)
<b>Note:</b> Debug Rev A only had the AND condition available for the triggers.</td></tr>
<tr><td>14
L1T</td><td>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)
<b>Note:</b> Debug Rev A only had the AND condition available for the triggers.</td></tr>
<tr><td>13
L1EBL</td><td>Enable Level 1 Breakpoint. Global enable for the breakpoint trigger.
0 Disables all level 1 breakpoints
1 Enables all level 1 breakpoint triggers</td></tr>
</table>

Table 36-8. TDR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>12–6
L1ED</td><td>Enable Level 1 Data Breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor’s local data bus. Clearing all L1ED bits disables data breakpoints.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>12</td><td>Data longword. Entire processor’s local data bus.</td></tr>
<tr><td>11</td><td>Lower data word.</td></tr>
<tr><td>10</td><td>Upper data word.</td></tr>
<tr><td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr>
<tr><td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr>
<tr><td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr>
<tr><td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr>
</table></td></tr>
<tr><td>5
L1DI</td><td>Level 1 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.
0 No inversion
1 Invert data breakpoint comparators.</td></tr>
<tr><td>4–2
L1EA</td><td>Enable Level 1 Address Breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.
<table>
<tr><th>TDR Bit</th><th>Description</th></tr>
<tr><td>4</td><td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr>
<tr><td>3</td><td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr>
<tr><td>2</td><td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr>
</table></td></tr>
<tr><td>1
L1EPC</td><td>Enable Level 1 PC breakpoint.
0 Disable PC breakpoint
1 Enable PC breakpoint</td></tr>
<tr><td>0
L1PCI</td><td>Level 1 PC Breakpoint Invert.
0 The PC breakpoint is defined within the region defined by PBR<i>n</i> and PBMR.
1 The PC breakpoint is defined outside the region defined by PBR<i>n</i> and PBMR.</td></tr>
</table>

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”.

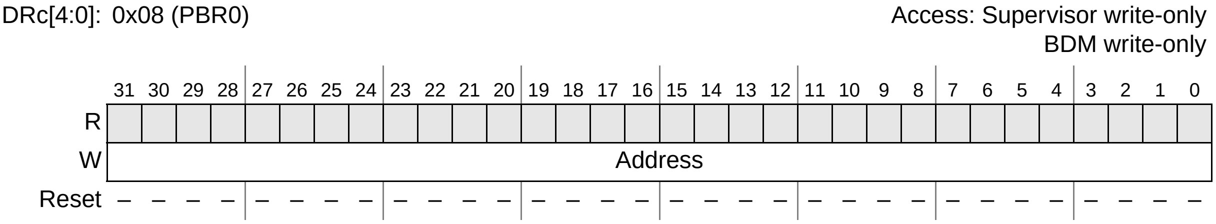


Figure 36-6. PC Breakpoint Register (PBR0)

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.

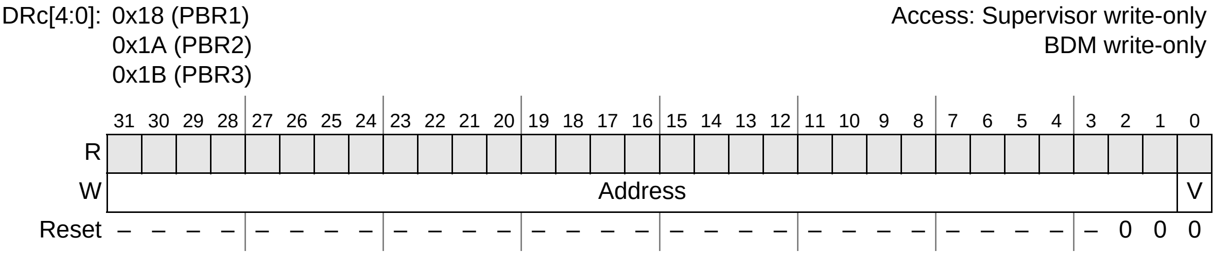


Figure 36-7. PC Breakpoint Register *n* (PBR*n*)

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

	31	30	29	28	27	26	25	24	23	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	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

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

	31	30	29	28	27	26	25	24	23	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 36-11. 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. Table 36-16 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 $\overline{\text{BKPT}}$ input is treated as a pseudo-interrupt; asserting $\overline{\text{BKPT}}$ creates a pending halt postponed until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction; if a pending halt is detected, the processor suspends execution and enters the halted state.

The are two special cases involving the assertion of $\overline{\text{BKPT}}$:

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the $\overline{\text{BKPT}}$ input is asserted within eight cycles after $\overline{\text{RESET}}$ is negated, the processor enters the halt state, signaling halt status (0xF) on the 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 $\overline{\text{BKPT}}$ asserted while the processor is stopped by execution of the STOP instruction. For this case, the processor exits the stopped mode and enters the halted state, at which point all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction, which follows the STOP opcode.

The CSR[27–24] bits indicate the halt source, showing the highest priority source for multiple halt conditions.

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.

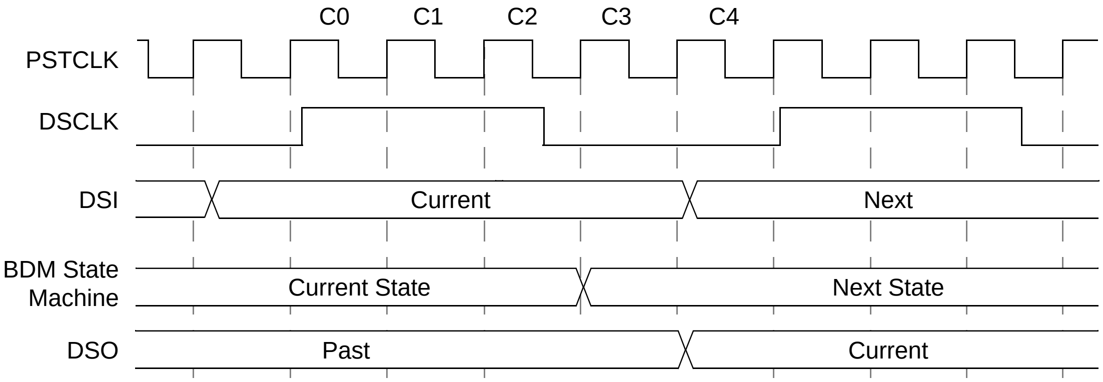


Figure 36-12. Maximum BDM Serial Interface Timing

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

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	Data															

Figure 36-13. Receive BDM Packet

Table 36-17. Receive BDM Packet Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16
S</td><td>Status. Indicates the status of CPU-generated messages listed below. The not-ready response can be ignored unless a memory-referencing cycle is in progress. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.
<table>
<tr><th>S</th><th>Data</th><th>Message</th></tr>
<tr><td>0</td><td>xxxx</td><td>Valid data transfer</td></tr>
<tr><td>0</td><td>FFFF</td><td>Status OK</td></tr>
<tr><td>1</td><td>0000</td><td>Not ready with response; come again</td></tr>
<tr><td>1</td><td>0001</td><td>Error-Terminated bus cycle; data invalid</td></tr>
<tr><td>1</td><td>FFFF</td><td>Illegal Command</td></tr>
</table>
</td></tr>
<tr><td>15–0
Data</td><td>Data. Contains the message to be sent from the debug module to the development system. The response message is always a single word, with the data field encoded as shown above.</td></tr>
</table>

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

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>15–10
Operation</td><td>Specifies the command. These values are listed in Table 36-20.</td></tr>
<tr><td>9</td><td>Reserved, must be cleared.</td></tr>
<tr><td>8
R/W</td><td>Direction of operand transfer.
0 Data is written to the CPU or to memory from the development system.
1 The transfer is from the CPU to the development system.</td></tr>
<tr><td>7–6
Op Size</td><td>Operand Data Size for Sized Operations. Addresses are expressed as 32-bit absolute values. A command performing a byte-sized memory read leaves the upper 8 bits of the response data undefined. Referenced data is returned in the lower 8 bits of the response.
<table>
<tr><th></th><th>Operand Size</th><th>Bit Values</th></tr>
<tr><td>00</td><td>Byte</td><td>8 bits</td></tr>
<tr><td>01</td><td>Word</td><td>16 bits</td></tr>
<tr><td>10</td><td>Longword</td><td>32 bits</td></tr>
<tr><td>11</td><td>Reserved</td><td>—</td></tr>
</table>
</td></tr>
<tr><td>5–4</td><td>Reserved, must be cleared.</td></tr>
<tr><td>3
A/D</td><td>Address/Data. Determines whether the register field specifies a data or address register.
0 Data register.
1 Address register.</td></tr>
<tr><td>2–0
Register</td><td>Contains the register number in commands that operate on processor registers. See Table 36-21.</td></tr>
</table>

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.

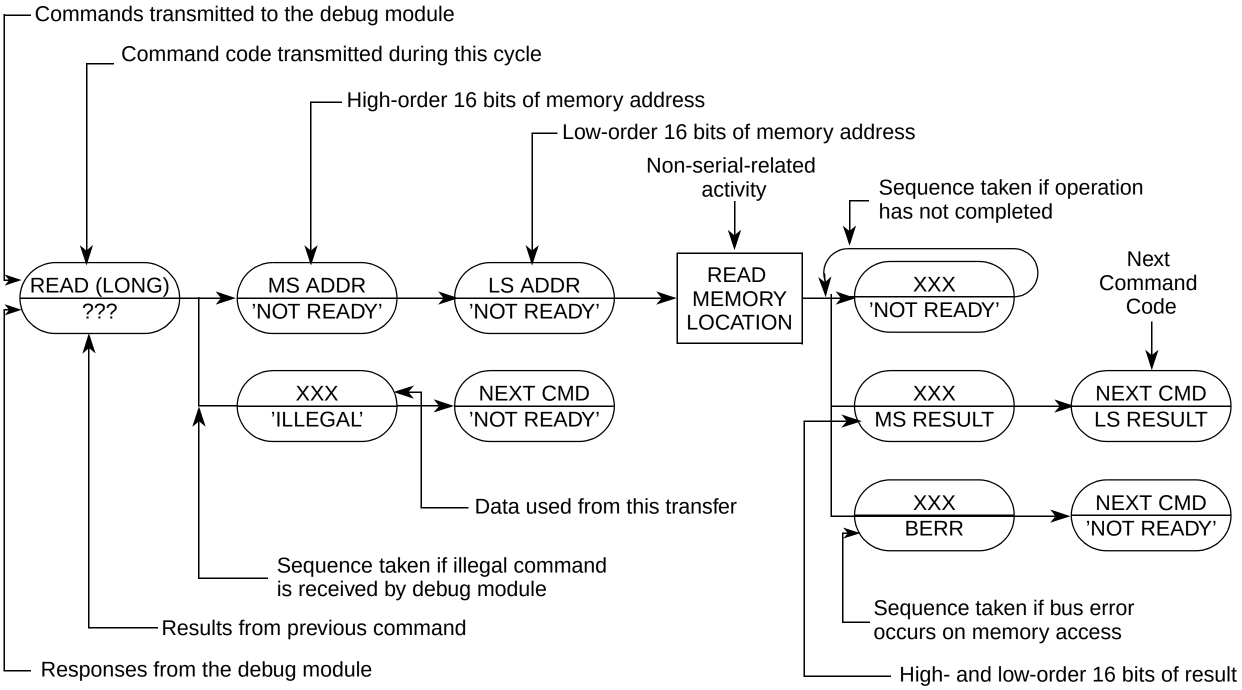

Commands transmitted to the debug module
Command code transmitted during this cycle
High-order 16 bits of memory address
Low-order 16 bits of memory address
Non-serial-related activity
Sequence taken if operation has not completed
READ (LONG)
???
MS ADDR
'NOT READY'
LS ADDR
'NOT READY'
READ MEMORY LOCATION
XXX
'NOT READY'
Next Command Code
XXX
'ILLEGAL'
NEXT CMD
'NOT READY'
XXX
MS RESULT
NEXT CMD
LS RESULT
XXX
BERR
NEXT CMD
'NOT READY'
Data used from this transfer
Sequence taken if illegal command is received by debug module
Results from previous command
Responses from the debug module
Sequence taken if bus error occurs on memory access
High- and low-order 16 bits of result

Figure 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:

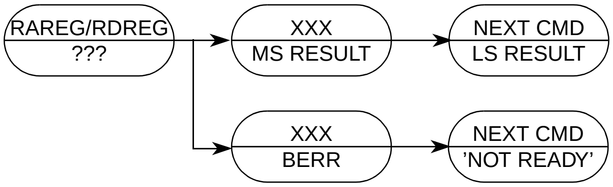


Figure 36-18. RAREG/RDREG Command Sequence

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:

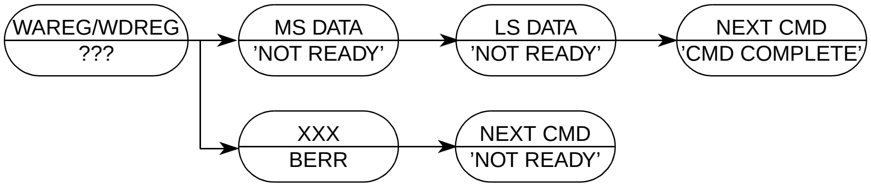


Figure 36-20. WAREG/WDREG Command Sequence

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:

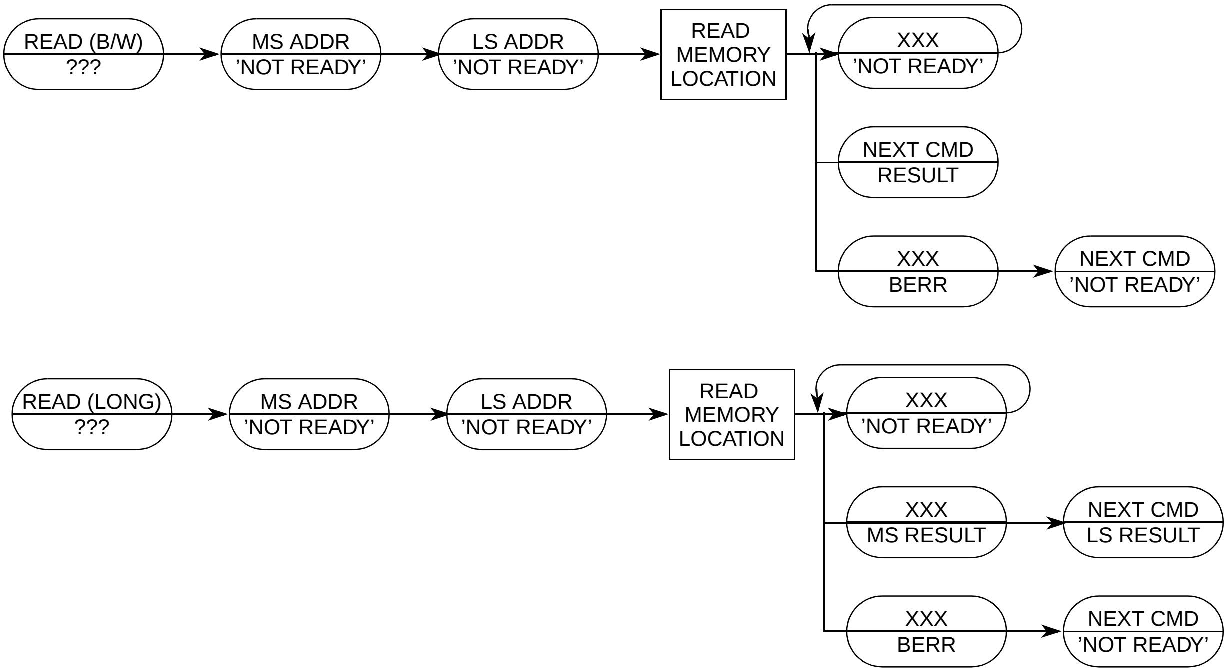


Figure 36-22. READ Command Sequence

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:

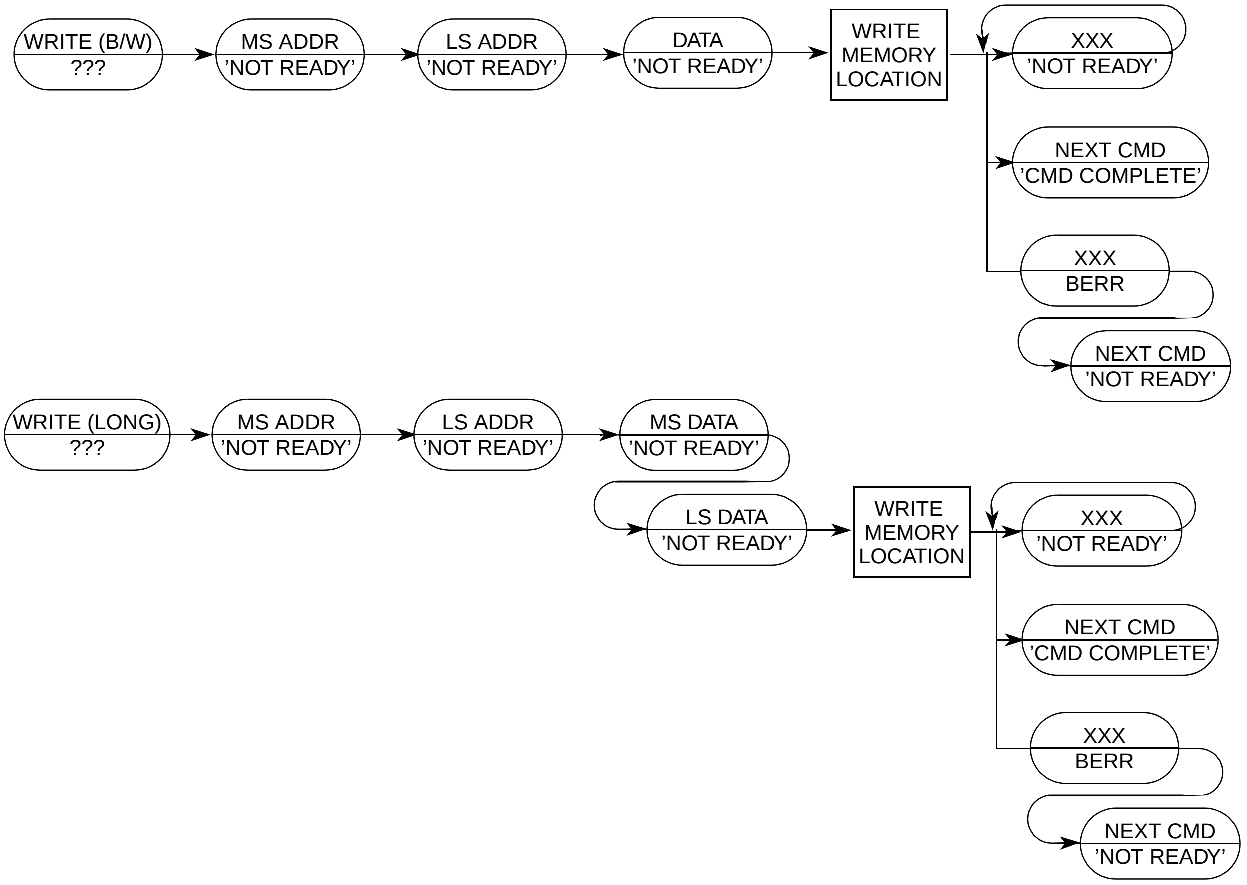


Figure 36-24. WRITE Command Sequence

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:

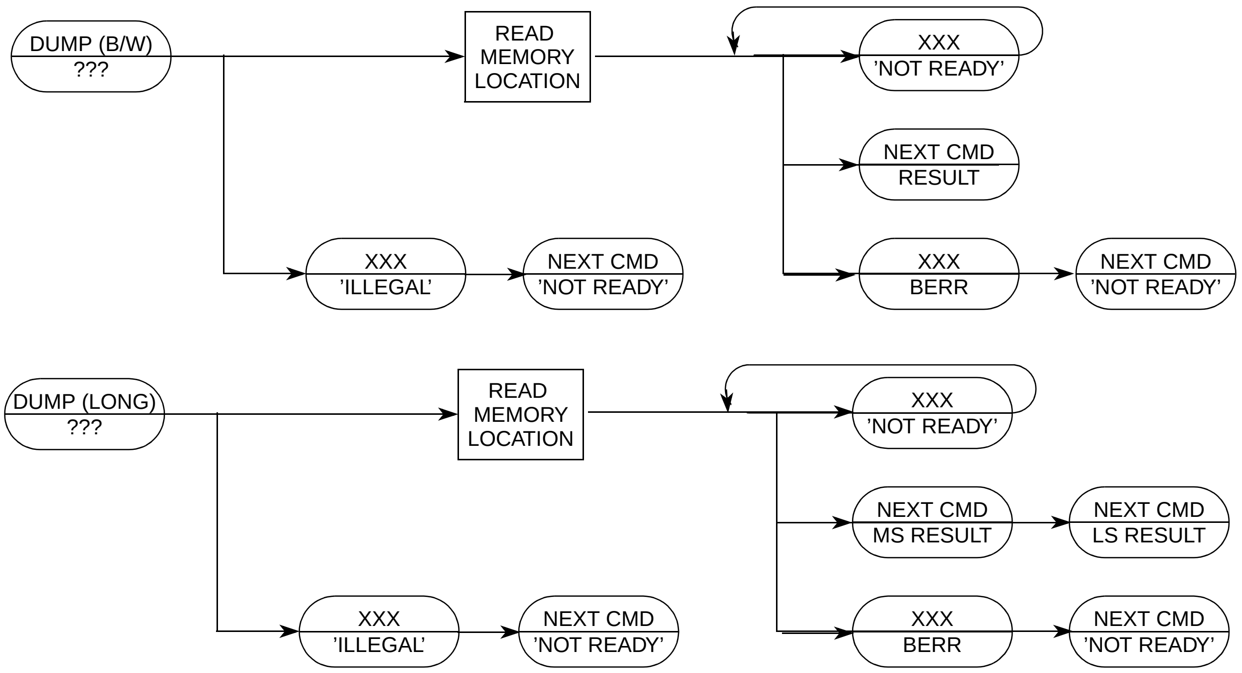


Figure 36-26. DUMP Command Sequence

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

Command Sequence:

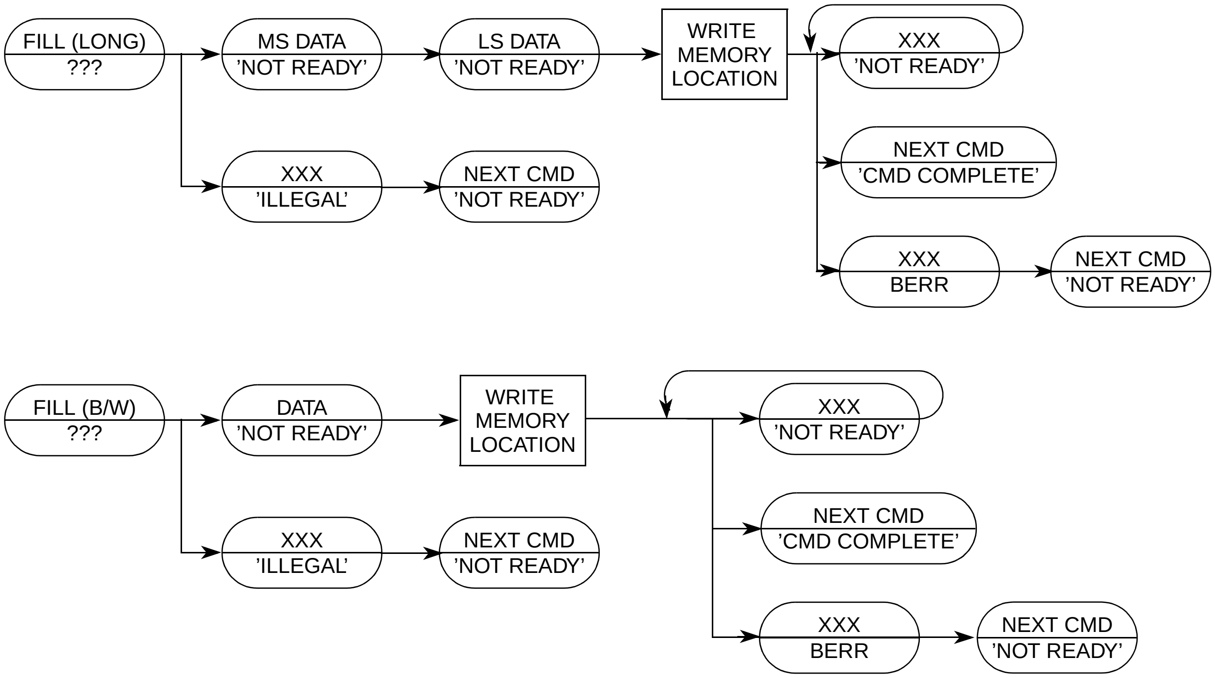


Figure 36-28. FILL Command Sequence

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:

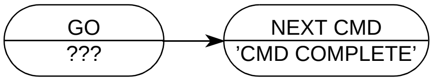


Figure 36-30. GO Command Sequence

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:

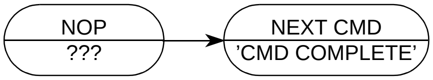


Figure 36-32. NOP Command Sequence

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:

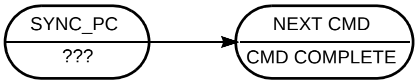


Figure 36-34. SYNC_PC Command Sequence

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:

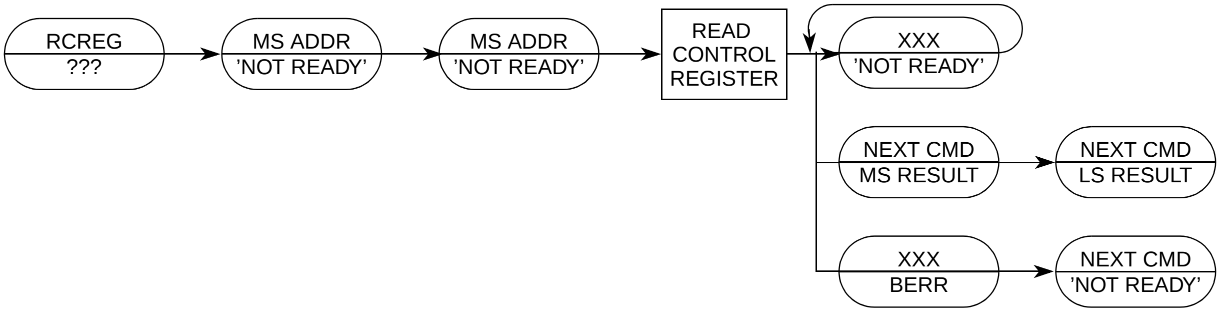


Figure 36-36. RCREG Command Sequence

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

Command Sequence:

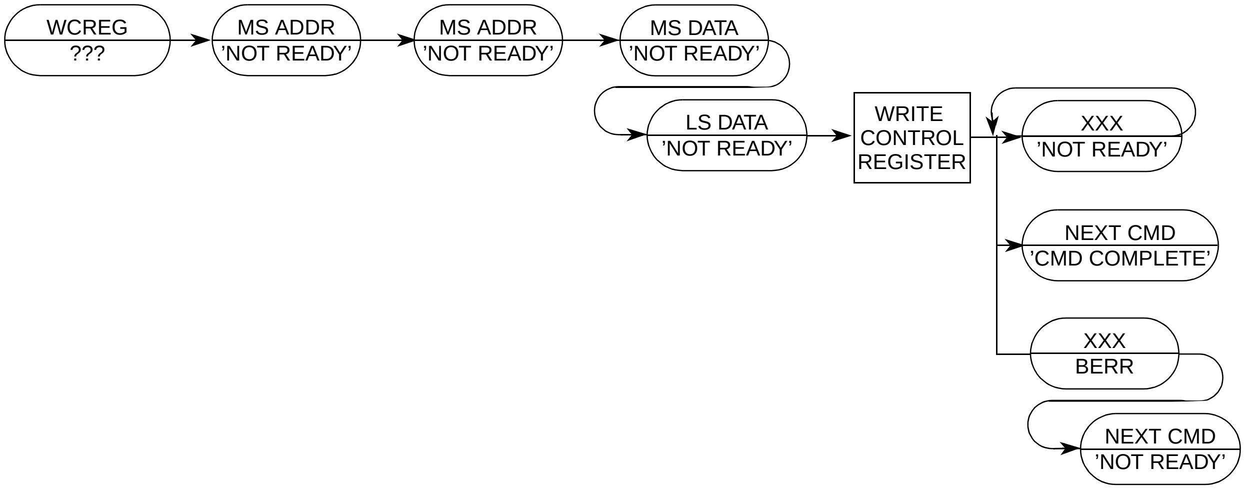


Figure 36-38. WCREG Command Sequence

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 (DRc=0x00).

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0xD				1	0	DRc					
Result	D[31:16]															
	D[15:0]															

Figure 36-39. RDMREG Command/Result Formats

Table 36-22 shows the definition of DRc encoding.

Table 36-22. Definition of DRc Encoding—Read

DRc[5:0]	Debug Register Definition	Mnemonic
0x00	Configuration/Status	CSR

Command Sequence:

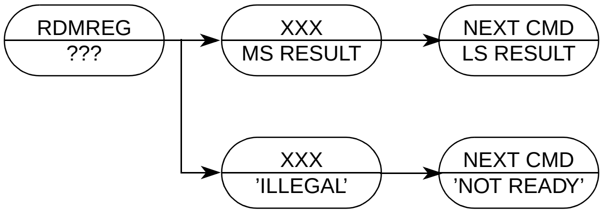


Figure 36-40. RDMREG Command Sequence

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				1	0	DRc					
D[31:16]															
D[15:0]															

Table 36-3 shows the definition of the DRc write encoding.

Command Sequence:

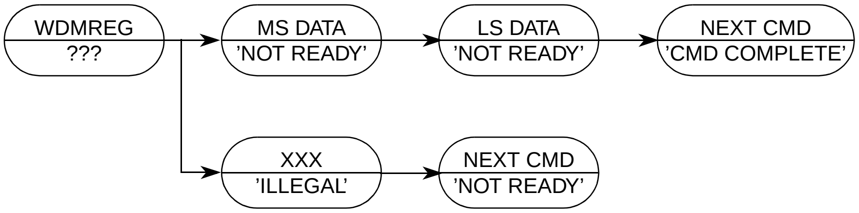


Figure 36-42. WDMREG Command Sequence

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.

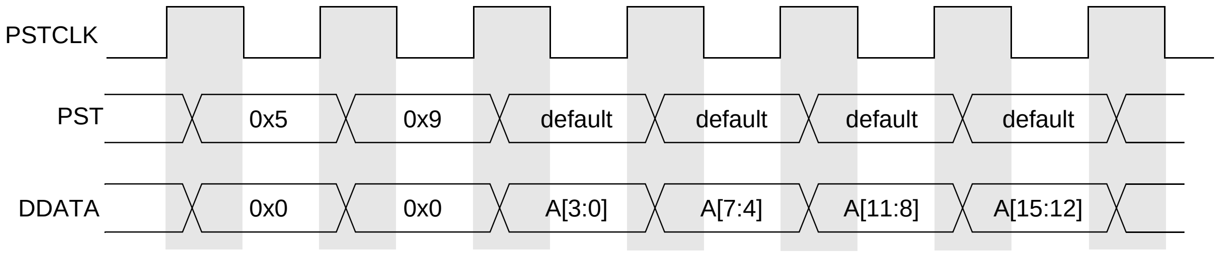


Figure 36-43. Example JMP Instruction Output on PST/DDATA

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.

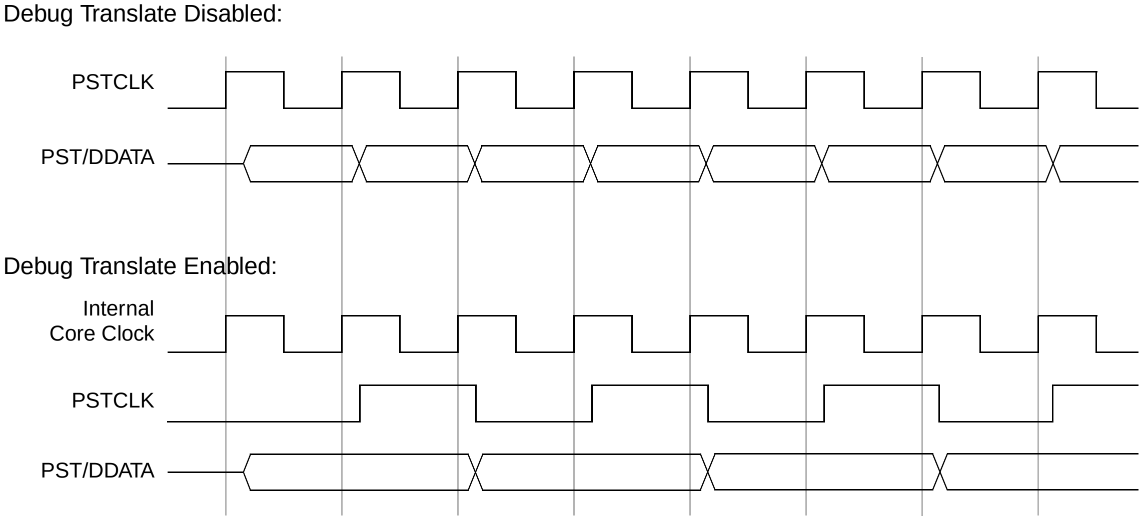


Figure 36-44. Debug Translate Timing

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 = 0x[9AB], DD = target address}

rts (not predicted)		PSTDDATA = 0x1, {0xB, source operand}, 0x5, {0x[9AB], target address}
rts (predicted)[4]		PSTDDATA = 0x1, {0xB, source operand}, 0x5
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}

Table 36-25. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
subx.l	Dy,Dx	PST = 0x1
swap.w	Dx	PST = 0x1
tpf		PST = 0x1
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.

[4] For a predicted RTS instruction, the source operand is displayed if CSR[12], CSR[9], or CSR[8] is set.

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}
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}

Table 36-27. PST/DDATA Specification for Supervisor-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
movec.l	Ry,Rc	PST = 0x1

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

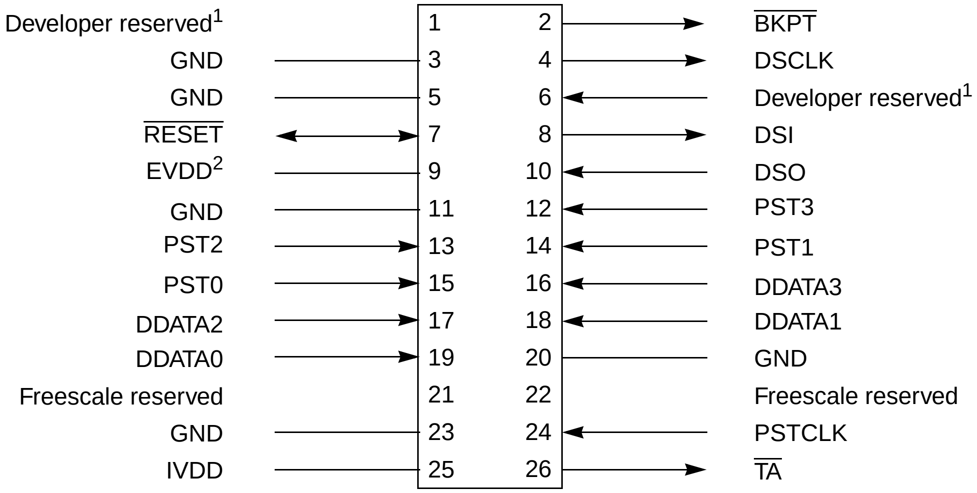
wdebug.l	<ea>y	PST = 0x1, {PST = 0xB, DD = source, PST = 0xB, DD = source}

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, $\overline{\text{TRST}}$.

37.1.1 Block Diagram

Figure 37-1 shows the block diagram of the JTAG module.

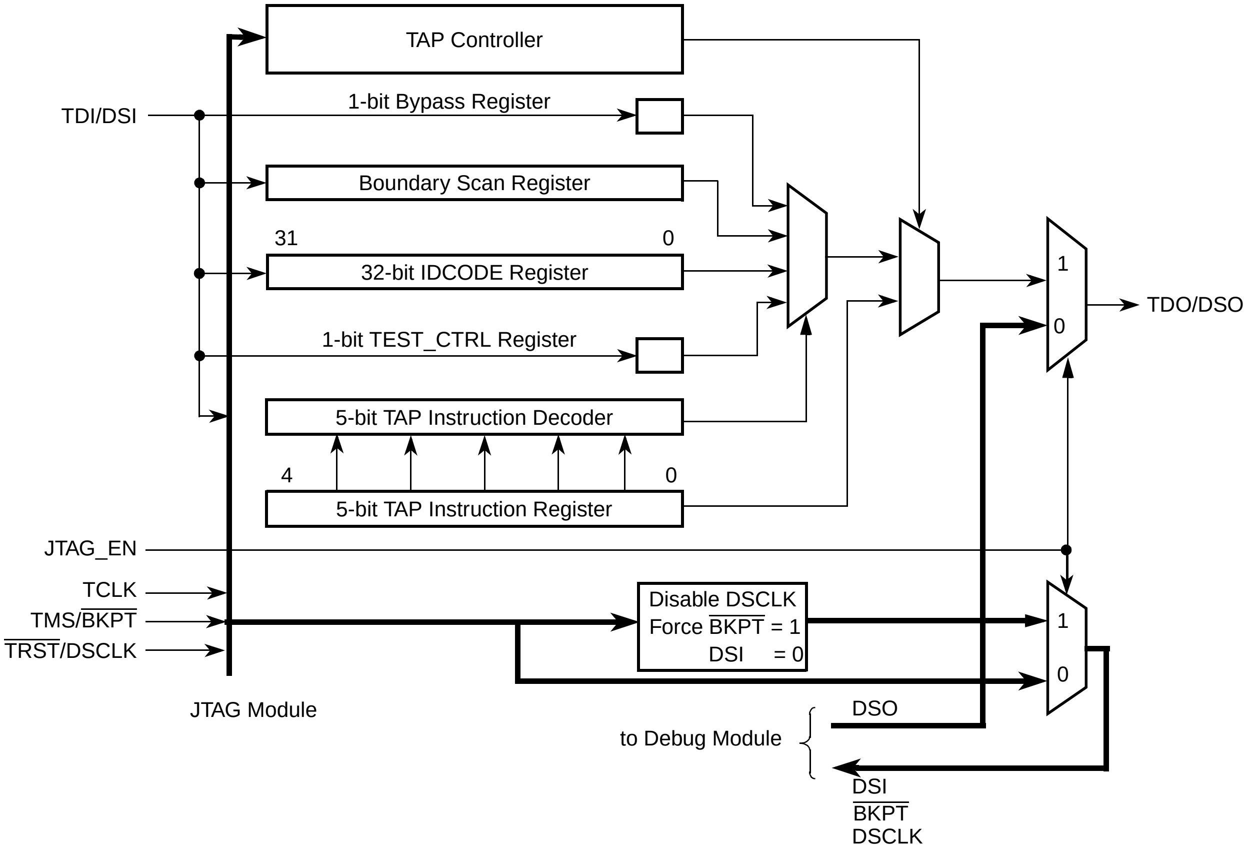


Figure 37-1. JTAG Block Diagram

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 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. 0x073 MCF53014 0x074 MCF53010 0x075 MCF53011 0x076 MCF53015 0x077 MCF53016 0x078 MCF53012 0x07C MCF53017 0x080 MCF53013
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.

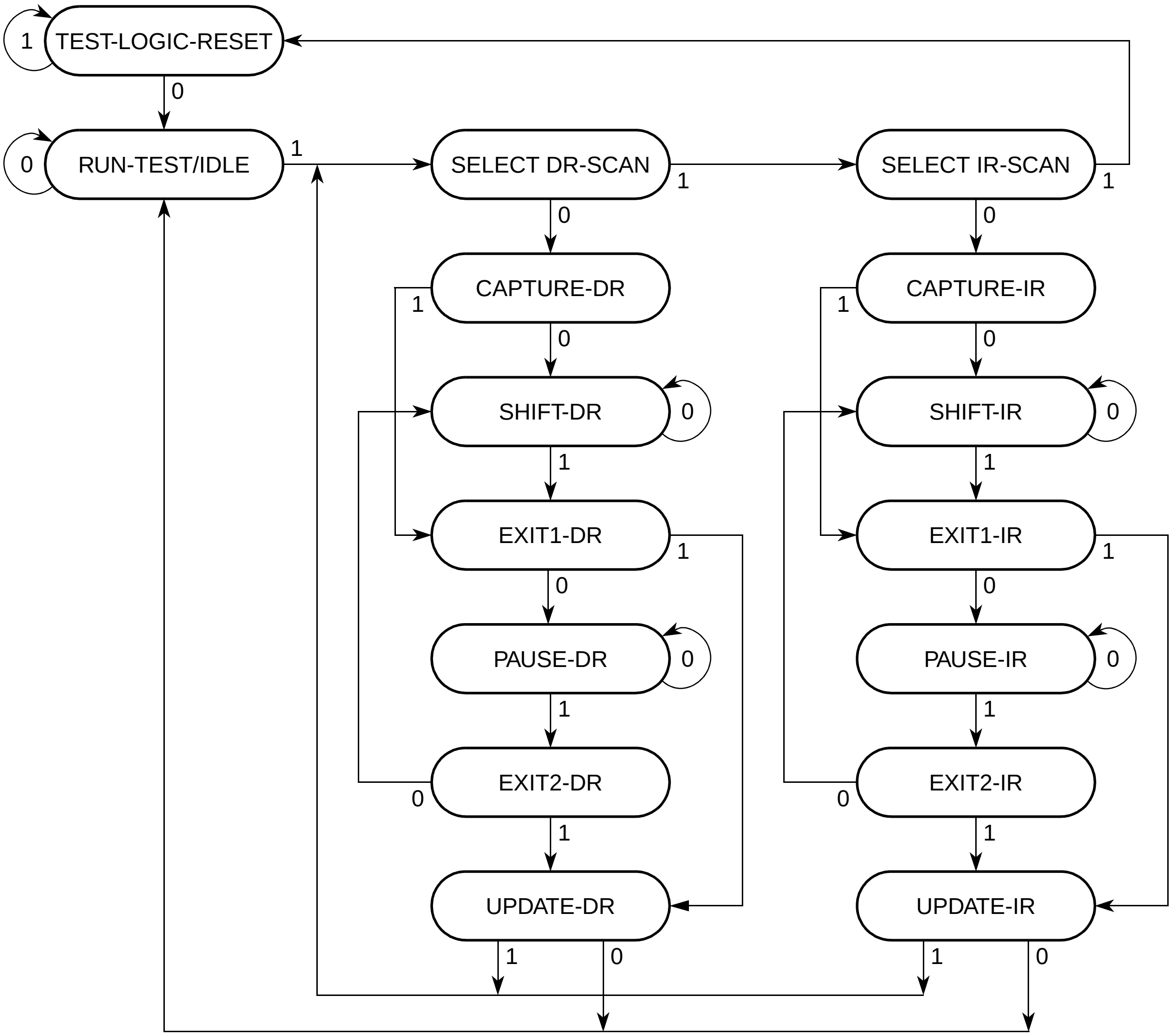


Figure 37-5. TAP Controller State Machine Flow

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 $\mathrm{EV}_{\mathrm{DD}}$.
- The TMS, TDI, and $\overline{\text{TRST}}$ pins include on-chip pull-up resistors. For minimal power consumption in low-power stop mode, these three pins should be connected to $\mathrm{EV}_{\mathrm{DD}}$ 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 $\overline{\text{TRST}}$ be pulled up. $\overline{\text{TRST}}$ could be connected to ground. However, because there is a pull-up on $\overline{\text{TRST}}$, some amount of current results. The internal power-on reset input initializes the TAP controller to the test-logic-reset state on power-up without asserting $\overline{\text{TRST}}$.